Training Text

マイコントレーニングボード
MT-R300

LCD基板（MT-E502）編

VPort Lab. 代表
元長野県工科短期大学校教授
工学博士 星野　俊行　著

Sunhayato

安全上のご注意

このたびは弊社製品をご使用いただき、誠にありがとうございます。本項では、誤った取り扱いによる事故を未然に防ぐための安全上の注意事項を説明しています。弊社製品をご使用になる前に必ずお読みください。

⚠警告	この表記を無視して誤った取り扱いをすると、死亡や重傷など、人体への重大な障害をもたらす恐れのある内容について示しています。
⚠注意	この表記を無視して誤った取り扱いをすると、軽傷または中程度の障害をもたらす恐れのある内容について示しています。また、本製品や本製品に接続している機器に損傷を与える可能性がある事項についても示しています。

⚠警告

水場禁止	▶水分の多いところ、水がかかる場所では、本製品は使用しないでください 風呂場や台所など水分の多いところ、水がかかる場所では、本製品は使用しないでください。 火災、感電、故障の原因となります。
禁止	▶医療、軍事、航空宇宙、列車、運送、原子力などの制御設備へは使用しないでください 医療機器、軍事機器、航空宇宙機器、運送、原子力などの制御設備などの人命に関わるシステムへの使用は意図しておりません。
接触禁止	▶雷が鳴りはじめたらご使用をお控えください 近くに雷が発生したときは、パソコンから本製品を抜いてご使用をお控えください。また、ご使用のパソコンの取扱説明書に従って、電源プラグをコンセントから抜くなどしてください。雷によっては、火災、感電、故障の原因となることがあります。

⚠注意

プラグの差し込み	▶プラグは確実に差し込んでください 差し込みが不完全ですと火災、感電、過熱、故障の原因になります。
発火注意	▶発火、発煙、異臭への対処 発火、発煙、異臭がするなどの異常がありましたら使用を直ちに中止してください。そのまま使用すると、火災、故障の原因となります。 すぐにパソコンから抜き、煙などの異常が出なくなるのを確認し、販売店などに修理をご依頼ください。
分解禁止	▶分解・改造しないでください 分解、改造しないでください。怪我、感電、故障の原因となります。本製品の分解、改造による怪我や事故について、当社は責任を負いかねます。
接触禁止	▶濡れた手での操作は避けてください 濡れた手で電源ケーブル・プラグを抜き差ししないでください。また、製品に触れないでください。感電の原因となることがあります。

⚠注意

禁止	▶以下のような場所では使用しないでください 本製品を以下のような場所で使用すると、動作不良、故障の原因となります。 ・振動や衝撃が加わる場所 ・直射日光のあたる場所 ・湿気やホコリが多い場所 ・温度差の激しい場所 ・熱を発生するもの（暖房器具など）の近く ・強い磁力、電波が発生するもの（磁石、ディスプレイ、スピーカー、ラジオ、無線機など）の近く ・湿気の多い場所
子供注意	▶子供の手の届かない場所に置いてください 本製品に装着されている電子部品など子供が飲み込まないように注意してください。
引抜禁止	▶通信中はケーブルを引抜かないでください 本製品がパソコンと通信中の場合はケーブルを引抜かないでください。ケーブルの引抜きは、必ず通信していないときに行ってください。故障の原因になることがあります。
安全設計	▶安全設計をしてください 本製品を、高度な信頼性を必要とするシステムに使用する場合は、冗長設計、誤動作防止設計など充分な安全設計を必ず行ってください。本製品の故障、傷害により生じるいかなる損害、事故について当社は責任を負いかねます。
保管注意	▶長期間使用しない場合の保管について 長期間使用しない場合は、帯電防止袋などに入れ、ホコリなどが入らないようにしてください。ホコリや汚れが付着すると短絡、接触不良などの原因になります。
ホコリ注意	▶製品の清掃について 製品にホコリや汚れなどが付着すると短絡、故障の原因になりますので、下記の「▶お手入れについて」に従って清掃してください。
薬品注意	▶お手入れについて ホコリが付着した場合はサンハヤト製ジェットブロー（JBK-480）などのガススプレーで吹き飛ばしてください。
使用注意	▶故障、破損時の処理について 本製品が故障もしくは破損した場合は、速やかに使用を中止してください。そのまま使用しますと火災、感電、怪我の原因になるおそれがあります。
廃棄注意	▶本製品の廃棄について 本製品の廃棄は、各自治体の廃棄ルールに従ってください。詳しくは各自治体にお問い合わせください。

本資料についてのご注意

本資料について

・本資料は、電子工作や電子回路、パーソナルコンピュータの操作について一般的な知識をお持ちの方を対象にしています。
・本資料を元に操作するには、株式会社ルネサス エレクトロニクス製 H8/300H マイコンについての知識や開発環境などが必要です。
・Microsoft®、Windows® は米国 Microsoft 社の米国およびその他の国における登録商標です。
・その他、記載されている会社名、製品名は各社の商標または登録商標です。

本資料のご利用にあたって

・この取扱説明書に掲載している内容は、お客様が用途に応じた適切な製品をご購入頂くことを目的としています。その使用により当社及び第三者の知的財産権その他の権利に対する保証、又は実施権の許諾を意味するものではありません。また、権利の侵害に関して当社は責任を負いません。
・本資料に記載した情報を流用する場合は、お客様のシステム全体で充分評価し適用可能かご判断願います。当社では適用可能判断についての責任は負いません。
・本資料に記載してある内容は、一般的な電子機器（学習教材、事務機器、計測機器、パーソナル機器、コンピュータ機器など）に使用されることを目的としています。高い品質や信頼性が要求され、故障や誤作動が直接人命を脅かしたり人体に危害を及ぼす恐れのある、医療、軍事、航空宇宙、原子力制御、運輸、移動体、各種安全装置などの機器への使用は意図も保証もしておりません。
・この取扱説明書の一部、又は全部を著者および当社の承諾なしで、いかなる形でも転載又は複製されることは堅くお断りします。
・全ての情報は本資料発行時点のものであり、当社は予告なしに本資料に記載した内容を変更することがあります。
・この資料の内容は慎重に制作しておりますが、万一記述誤りによってお客様に損害が生じても当社はその責任を負いません。
・本資料に関してのお問合せ、その他お気付きの点がございましたら、当社までお問合せください。
・本資料に関する最新の情報はサンハヤト株式会社ホームページ（http://www.sunhayato.co.jp/）に掲載しております。

このテキストについて

本テキストでは、サンハヤト H8 マイコントレーニングボード MT-R300 のプログラム開発を、HEW（High-Performance Embedded Workshop）を使用して行った場合について説明しています。HEW の詳細な使用方法や注意事項につきましては、ルネサス エレクトロニクス社発行のマニュアルを参照してください。

サンハヤト H8 マイコントレーニングボード MT-R300 概要

- ターゲットマイコン　：H8/300H シリーズ 3062 グループ HD64H3062BF（以下 H8/3062BF マイコン）
- 書き込みボード　　：MT-R300（H8/3062BF マイコン モード 5（内蔵 ROM 有効拡張 16M バイトモード）、モード 7（シングルチップアドバンストモード）対応）

開発に必要なもの

- 開発用 PC（以下の条件を満たすもの）

PC 本体	Pentium Ⅲ 600MHz 以上を搭載した IBM PC/AT 互換機
OS	Windows XP, Windows Me, Windows 98SE, Windows 2000（Windows XP 推奨）
メモリ	128MB 以上（ロードモジュールの 2 倍以上）
ハードディスク	エミュレータ用ソフトウェアのインストールには 100MB 以上の空き容量が必要
入力デバイス	マウス、またはマウス相当のポインティングデバイス
インターフェイス	1 ポート以上の USB インターフェイス

- サンハヤト H8 マイコントレーニングボード　MT-R300
- 付属の USB ケーブル

このマニュアルで使用しているツール類

このスタートアップガイドでは、以下の OS、ツールのバージョンで動作したものとして説明しています。

- ホスト PC の OS　　：Windows XP Professional
- 統合化開発環境　　：HEW V.4.03.00001
- C コンパイラ　　：H8S,H8/300 Standard Toolchain (V.6.02 Release00)
- フラッシュ書き込みツール　：FDT（FlashDevelopmentToolKit ）Ver4.02

HEW のバージョンは、HEW の「 ヘルプ 」メニューの「High-Performance Embedded Workshop のバージョン情報 」で確認できます。

このマニュアルで紹介している各社サイト、ツールについて

本スタートアップガイドで紹介している各社サイトやツールは、本スタートアップガイド発行時のものを掲載しております。各社サイトの構成やツールのバージョン、またバージョンに伴ってツールのファイル名などが変わっている場合がありますのでご了承ください。

目　次

このテキストについて …… 5

サンハヤト H8 マイコントレーニングボード MT-R300 概要 …… 5
開発に必要なもの …… 5
このマニュアルで使用しているツール類 …… 5
このマニュアルで紹介している各社サイト、ツールについて …… 5

1. はじめに …… 7

2. 基礎知識 …… 8

2.1　液晶 …… 8
2.2　LCD モジュールの制御方法 …… 9

3. 接続図の概要 …… 15

4. 基本実習 …… 16

4.1　8 ビットモードでの制御 …… 16
実習 4.1.1　文字を表示する（DDRAM へのデータ書き込み） …… 16
実習 4.1.2　ビジーチェックプログラムを作成する …… 20
実習 4.1.3　LCD の 1 行目に文字を表示し、それを読み出し（演算を加え）、2 行目に表示する …… 22
4.2　4 ビットモードでの制御 …… 26
実習 4.2.1　文字を表示する（DDRAM へのデータ書き込み） …… 26
実習 4.2.2　ビジーチェックプログラムを作成する …… 29
実習 4.2.3a　LCD の 1 行目に文字を表示し、それを読み出し（演算を加え）、2 行目に表示する …… 32
実習 4.2.3b　実習 4.2.3a のプログラムの関数名を標準的な関数名に変更し、必要な関数を追加する …… 36
実習 4.2.4　ビープ音（1kHz）を発生する …… 40
4.3　ライブラリの作成（4 ビットモード） …… 41
実習 4.3.1a　1 行目を下方向（2 行目）にシフト（コピー）する。 …… 41
実習 4.1.3b　実習 4.3.1a を参考に、上下方向にシフトする関数を作成する …… 46
実習 4.3.2a　文字列を表示し、それを左右方向にローテーションする …… 47
実習 4.3.2b　実習 4.3.2a を参考に、左右方向にローテーションする関数を作成する …… 48
実習 4.3.3　LCD に整数（long 型）を表示する（引数：フィールド幅 , 整数） …… 50
実習 4.3.4　ビープ音を発生する …… 52

5. おわりに …… 54

1. はじめに

このボードは、液晶ディスプレイ（Liquid Crystal Display、以下「LCD」という）モジュールを搭載し、16 文字× 2 行のアルファベットの大文字、小文字、カタカナ、図形文字などを表示する教育用ボードです。これを用いて LCD モジュールのハードウエアの概要を学習し、インストラクションを読み書きする制御プログラムを作成します。さらにポートの余った 1 ビットを用いてブザーを鳴動させます。

この LCD モジュールは、インストラクションをそのまま読み書きする 8 ビットモード、それを 2 回に分けて読み書きする 4 ビットモード、のいずれかで動作します。8 ビットモードではデータバス線 8 本と制御信号線（リード / ライト信号、イネーブル信号、レジスタ選択信号）3 本の計 11 本（2 ポート）、4 ビットモードではデータバス線 4 本と制御信号線 3 本の計 7 本（1 ポート）、で制御します。

この LCD モジュールを搭載した類似の市販品や関連書籍中の参考回路図等を見ると、ハードウエアが 4 ビットモードに固定され、リード / ライト信号がライトに固定されているものが大半です。4 ビットモードの場合、ハードウエアは簡単ですがソフトウエア（制御プログラム）が複雑になるため初心者には向きません。リード / ライト信号がライトに固定されている場合、データや状態（ステータス）を読み出すことができません。

このボードは、ポートコネクタの選択で、8 ビットと 4 ビットの 2 モードを切り替えられます。これにより初心者は簡単な 8 ビットモードで制御プログラムを作成し、十分理解を深めてから複雑な 4 ビットモードに移行できます。リード時には、データだけでなく、状態（ビジー信号など）も取り込めるため、書き込み速度向上のためのビジーチェックシーケンスについて学習できます。さらに、制御信号やデータバス信号のレベル（H/L）をモニターする LED を設置しているため、ゆっくり動作させることで各信号の変化（タイミング）を視認できます。従って、このボードを用いることでより効果的な教育ができます。

2. 基礎知識

2.1 液晶

液晶とは、文字どおり液体状（Liquid）の結晶（Crystal）のことで、これを用いた表示装置（Display）がLCDです。図2-1はTN型の液晶分子の整列状態で、（a）は通常時、（b）は電界をかけた時を示します。微細な溝のある配向板に液晶を乗せると、液晶分子は溝に沿って整列します。溝の向きが直交する2枚の配向板で液晶を挟むと 、両板の溝に沿って分子が90度ねじれて整列するため、その間に挟まれた分子は緩やかに90度ねじれて整列します。電界をかけた時は、分子は電界の方向に整列します。電界は、液晶（配向板を含む）を挟む2枚の透明電極に電圧をかけて作ります。液晶の後面から光が入射すると、その光は液晶分子の並び（隙間）に沿って進むため、出射光は通常時には90度ねじれ、電界をかけた時には直進します。

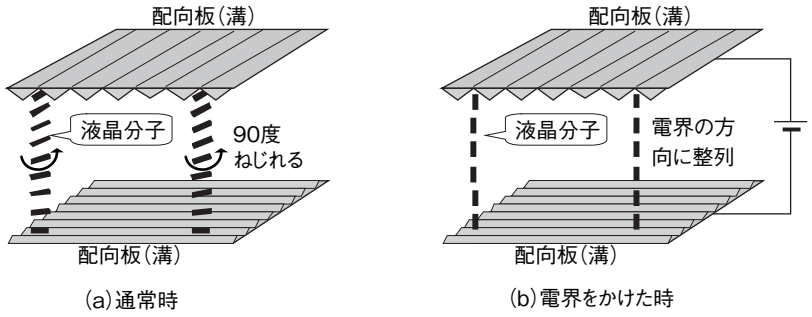


図2-1　液晶分子の整列状態（TN型の例）

図2-2は液晶パネルの原理で、（a）は通常時、（b）は電界をかけた時を示します。液晶パネルは、偏光方向を直交させた2枚の偏光フィルターで液晶層を挟む構造になっています。背面からの光が後側の偏光フィルター（垂直方向）を通過し、液晶層に入射します。通常時は、液晶層で90度ねじれて水平方向成分に変化し出射するので、前側の偏光フィルター（水平方向）を通過します（明るい）。電界をかけた時は、液晶層でのねじれがなくなり垂直方向成分のまま出射するので、前側のフィルター（水平方向）で遮断されます（暗い）。背面からの光は、外光やバックライト等の光源を用います。なお、2枚の偏光フィルターの偏光方向をそろえて、光の通過と遮断の動作を逆にしたタイプもあります。

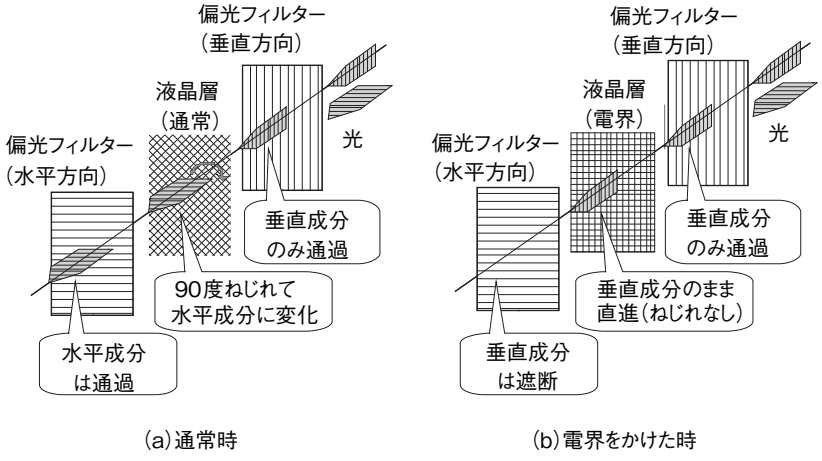


図2-2　液晶パネルの原理

液晶パネルは、多数の画素を格子状に配列したドットマトリックスにより構成されています。縦横方向の2次元的な配線の交点（画素）を制御するマトリックス配線方式が一般的です。この場合の配線数は、縦線と横線の合計数で済みます。これにより各画素の電圧を個別に制御し、光の通過量（透過率）を変化させて、パネル面に明暗のパターン（文字、画像）を表示します。

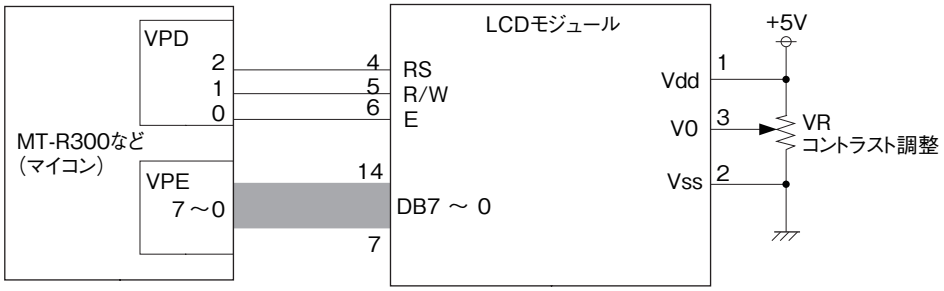
LCDモジュールとは、一般的にこの液晶パネルにLCDコントローラ、セグメントドライバー回路、電源回路、コネクタなどを付加したものを指します。

2.2　LCD モジュールの制御方法

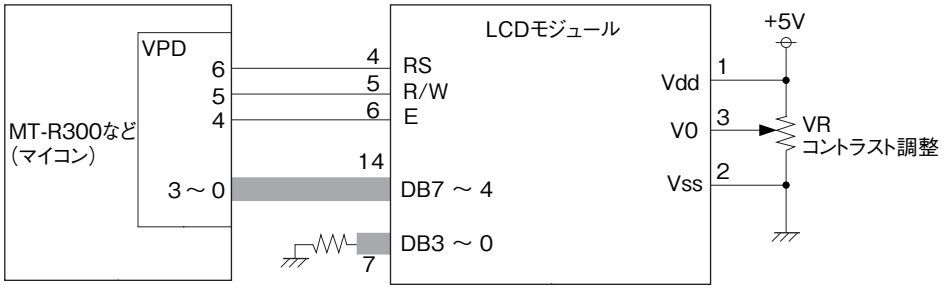
(1) マイコンとの接続

このボードに搭載されている LCD モジュールは、リビジョン番号により異なります。リビジョン番号の表記がない（以下便宜的に「Rev.1.0」という）ボードは Sunlink Display Tech 社の製品で、Rev.2.0 のボードはサンエンジニアリング社の製品です。この 2 つはアクセスタイミングが異なるだけで、その他の仕様は全く同等です。

これらはパラレルインターフェースを介して、マイコンからのインストラクション（命令またはデータ）で制御されます。インストラクションの読み書き動作は 8 ビットモードと 4 ビットモードのいずれかで行います。なお、この内容は両社のデータシートから抜粋したもので、詳細はそちらを参照願います。サンハヤト社の LCD ボード MT-E502 取扱説明書も併せて参照願います。



(a)8ビットモード時



(b)4ビットモード時

図 2-3　パラレルインターフェースによる制御

図 2-3 はパラレルインターフェースによる制御方法で、(a) が 8 ビットモード時、(b) が 4 ビットモード時、を示します。8 ビットモードではマイコンのポート（8 ビット）に DB7 ～ 0 を接続し、インストラクションを 1 回で読み書きします。4 ビットモードではマイコンのポート（4 ビット）に DB7 ～ 4 を接続し、インストラクションを上位 4 ビット、下位 4 ビットの順に 2 回に分けて読み書きします。この場合、DB3 ～ 0 には特に信号を入力しなくても動作しますが、H/L レベルのいずれかに固定した方が望ましいので、ここでは L に固定します。8 ビットモードでの読み出し時には DB3 ～ 0 に H の信号が出力されショートする可能性があるため、プルダウン抵抗を挿入します。

(2) 読み書きのタイミング

図 2-4 は LCD モジュールにインストラクション（命令（読み込み時は状態）、データ）を読み書きする時の各信号のタイミングチャートを示します。2 社いずれの製品でも動作させるため、図中のタイミング時間は長い方を表示しています。

同図（a）は命令、(b) はデータ、の書き込み動作でその概要は次のとおりです。

① E（イネーブル）信号を 0（動作禁止）にします。

② RS（レジスタ選択）信号を 0（命令）または 1（データ）に、R/W（リード / ライト）信号を 0（書き込み）に設定します（①と同時でも良い）。
③ RW、RS セットアップ時間 40ns（Rev.1.0:40ns、Rev.2.0:0ns）以上待ちます。クロック周波数 25MHz、周期 1/（25 × 106）=40ns（1 クロック分）のため、実際には不要です。
④ E 信号を 1（動作許可）にします。
⑤ DB7 ～ 0 にインストラクションを書き込みます。
⑥動作許可時間 230ns（Rev.1.0:230ns、Rev.2.0:140ns）以上待ちます。
⑦ E 信号を 0（動作禁止）にします。実際にはこの立下り時にインストラクションが受信されます。
⑧①～⑧までのサイクル時間 1200ns（Rev.1.0:500ns、Rev.2.0:1200ns）の不足時間待ちます。不足時間は 1200 －（40 + 230 +（処理時間の合計））で計算します。なお、サイクル時間は通常イネーブル信号の立ち上がりから次の立ち上がりまでの時間ですが、ここでは基準点を変えて①～⑧までをサイクル時間としています。

同図（c）は状態、（d）はデータの読み出し動作で、その概要は次のとおりです。

① E 信号を 0（動作禁止）にします。
② RS 信号を 0（命令）または 1（データ）に、R/W 信号を 1（読み出し）に設定します（①と同時でも良い）。
③ RW、RS セットアップ時間 40ns（Rev.1.0:40ns、Rev.2.0:0ns）以上待ちます。クロック周波数 25MHz、周期 1/（25 × 106）=40ns（1 クロック分）のため、実際には不要です。
④ E 信号を 1（動作許可）にします。
⑤動作許可時間 230ns（Rev.1.0:230ns、Rev.2.0:140ns）以上待ちます。

図 2-4　インストラクション (命令（状態）、データ) の書き込み、読み出しのタイミングチャート

⑥ DB7 ～ 0 からインストラクションを読み出します。

⑦ E 信号を 0（動作禁止）にします。

⑧①～⑧までのサイクル時間 1200ns（Rev.1.0:500ns、Rev.2.0:1200ns）の不足時間待ちます。不足時間は 1200 －（40 ＋ 230 ＋（処理時間の合計））で計算します。

（3）文字コードと文字パターン

表 2-1 は文字コードと文字パターンの対応表です。この中の（0x20 ～ 0xdf）は、大文字、小文字、カタカナで、JIS X 0201 に準じています。（0xe0 ～ 0xff）は図形文字です。これらはあらかじめモジュール内の CGROM（Character Generator ROM）に格納されています。（0x00 ～ 0x07）はユーザーが独自に作成した 8 種類の独自図形文字で、自身で CGRAM（Character Generator RAM）に格納しておきます。それ以降は DDRAM（Display Data RAM）にそれらの文字コードを格納するだけで、自動的に LCD モジュールに対応する文字パターンを表示します。独自図形文字パターンの作成法はここでは省略します。

表 2-1　文字コードと文字パターンの対応表

		上位4ビット（D4～D7）HEX															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
下位4ビット（D0～D3）HEX	0	CG RAM (1)															
	1	CG RAM (2)															
	2	CG RAM (3)															
	3	CG RAM (4)															
	4	CG RAM (5)															
	5	CG RAM (6)															
	6	CG RAM (7)															
	7	CG RAM (8)															
	8	CG RAM (1)															
	9	CG RAM (2)															
	A	CG RAM (3)															
	B	CG RAM (4)															
	C	CG RAM (5)															
	D	CG RAM (6)															
	E	CG RAM (7)															
	F	CG RAM (8)															

(4) インストラクション

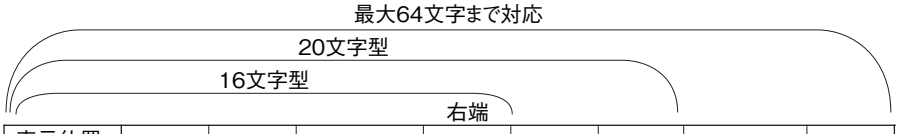
表 2-2 は、インストラクションの一覧を示します。RS（Register Select）は命令（読み出し時は状態）かデータを示します。R/W（Read/Write）は読み出しか書き込みを示します。それぞれの使い方は、各実習の中で解説します。

表 2-2　インストラクションの一覧表

インストラクション	コード										内容	実行時間 (max.)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
表示クリア	命令	書き込み	0	0	0	0	0	0	0	1	全表示をクリアする （カーソルはホーム位置に戻る）	1.53ms
リターンホーム	命令	書き込み	0	0	0	0	0	0	1	0	DDRAM アドレスに“00H”を設定し、カーソルをホーム位置に戻す （シフトしていた表示も元に戻る）	1.53ms
エントリーモードセット	命令	書き込み	0	0	0	0	0	1	I/D	S	I/D: カーソル移動（0: 左（AC--）、1: 右（AC++）） S: 表示シフト（0: しない、1: する） （データの書き込み、読み出し時）	39μs
表示オン / オフ	命令	書き込み	0	0	0	0	1	D	C	B	D: 全表示（0: オフ、1: オン） C: カーソルの表示（0: オフ、1: オン） B: 表示文字のブリンク（0: オフ、1: オン）	39μs
カーソル / 表示シフト	命令	書き込み	0	0	0	1	S/C	R/L	0	0	S/C:（0: カーソルシフト、1: 表示全体シフト） R/L:（0: 左、1: 右） （DDRAM の内容は変化しない）	39μs
ファンクションセット	命令	書き込み	0	0	1	DL	N	F	0	0	DL: データ長（0:4 ビット、1:8 ビット） N: デューティ（0:1 行、1:2 行（5 × 7 ドット）） F: フォント（0:5 × 7 ドット、1:5 × 10 ドット）	39μs
CGRAM アドレスセット	命令	書き込み	0	1	ACG（0x00 ～ 0x3f）						AC に ACG（CGRAM（Character Generator RAM）のアドレス）をセット	39μs
DDRAM アドレスセット	命令	書き込み	1	ADD（0x00 ～ 0x7f）							AC に ADD（DDRAM（Display Data RAM）のアドレス）をセット	39μs
BF/ アドレス読み出し	命令（状態）	読み出し	BF	AC（7 ビット）							ステータス（BF（ビジーフラグ）/AC（アドレスカウンタ））を読み出す	0μs
CGRAM、DDRAM への書き込み	データ	書き込み	書き込みデータ（8 ビット）								直近でアドレスセットされた RAM（DDRAM または CGRAM）にデータを書き込む	43μs
CGRAM、DDRAM からの読み出し	データ	読み出し	読み込みデータ（8 ビット）								直近でアドレスセットされた RAM（DDRAM または CGRAM）からデータを読み出す	43μs

(5) DDRAM のアドレス

図 2-5 は LCD モジュールの表示位置と DDRAM のアドレスの対応を示します。最大 64 文字に対応させているため、1 行目と 2 行目はアドレスが離れますので注意してください。アドレスが 0x00 ～ 0x7f のため、その長さは 7 ビットになります。



表示位置	1	2	～	16	～	20	～	64
1行目	0x00	0x01	～	0x0f	～	0x13	～	0x3f
2行目	0x40	0x41	～	0x4f	～	0x53	～	0x7f

図 2-5　LCD モジュールの表示位置と DDRAM のアドレスの対応

(6) 初期設定

図 2-6 は、LCD モジュールを動作させるための初期設定の手順で、(a) は 8 ビットモード時、(b) は 4 ビットモード時、を示します。その概要は次のとおりです。

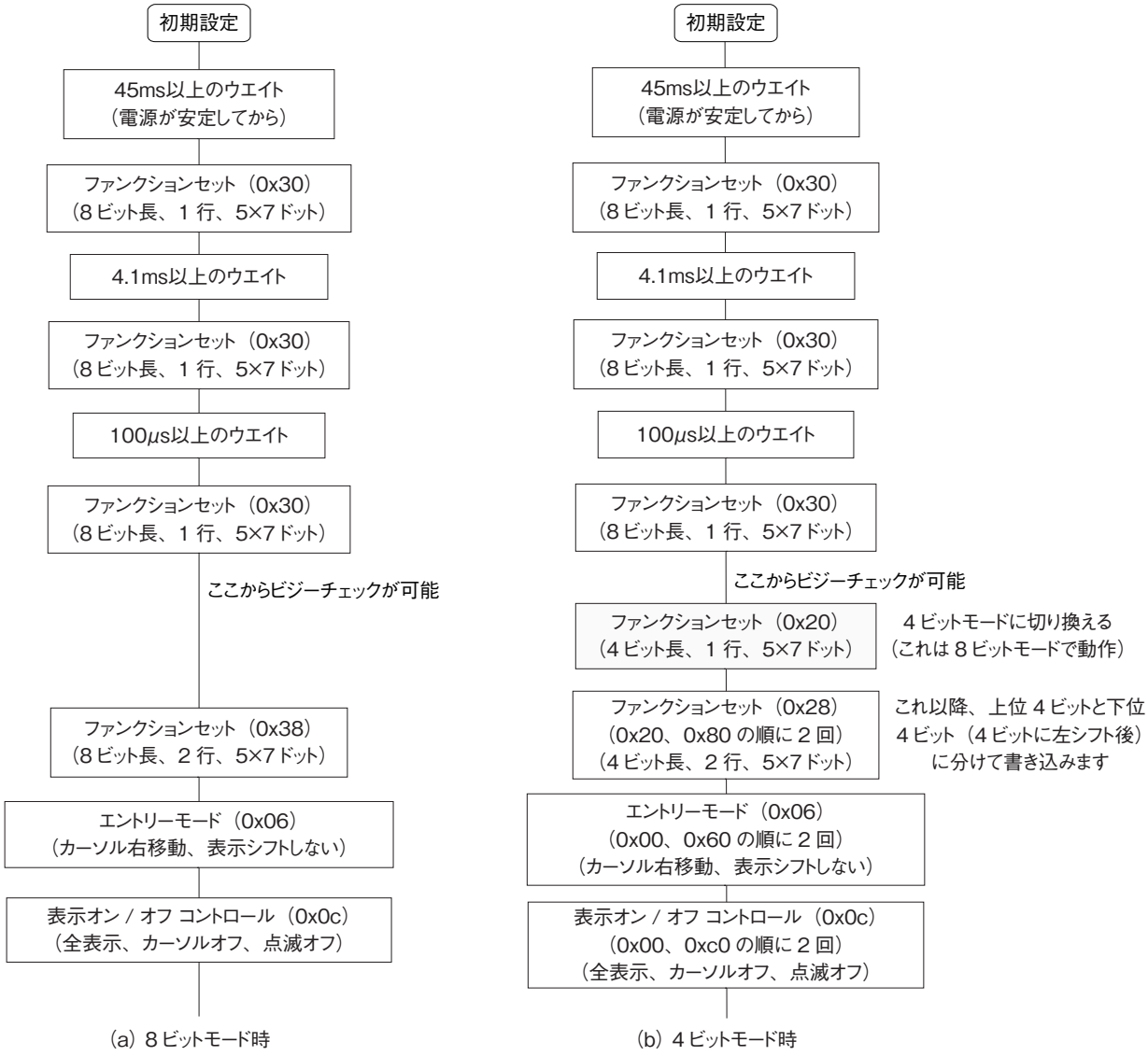


図 2-6　LCD モジュールの初期設定

①電源投入後、その電圧が安定する（+Vdd が 4.5V 以上になる）までの時間と、LCD モジュールの初期設定の時間（45ms）を待ちます。通常マイコンも同時に電源投入されるため前者の時間は省略できます。
②インターフェース長が何ビットに設定されているか分からないので、ファンクションセット（0x30）を 2 回以上（念のため 3 回）繰り返して、まず 8 ビット長に設定します。
③②の繰り返しの間で、必要なウエイト（4.1ms、100μs）をとります。
④ファンクションセットで、インターフェース長を適切なビット長に設定変更します（8 ビットモード時は設定済みですから省略します）。
⑤ファンクションセットで行数やフォントのドット数を適切に設定します。
⑥エントリーモードでカーソル移動方向や表示シフトを適切に設定します。
⑦表示オン / オフでカーソルの表示や点滅を適切に設定します。

4 ビットモードの場合、DB3 ～ 0 がハードウエアによりプルダウンされていますので、④までは設定値の下位 4 ビッ

トが0に固定されます。④以降ビジーチェックが可能になります。これを省略する場合は十分なウエイトをとります。⑤以降の設定で8ビットモード時は1回で書き込みますが、4ビットモード時は上位4ビットと下位4ビット（左に4ビットシフト後）に分けて2回書き込みします。

初期設定には表示クリアを入れていないので、必要に応じてプログラム内に配置します。

3. 接続図の概要

図 3-1 はこのボードの接続図の概要を示します。8 ビットモードでは VP2、VP3 の 2 つのポートを使います。4 ビットモードでは VP1 ポートのみを使います。　制御信号（R/W、E、RS）は 4 ビットモード時 VP1、8 ビットモード時 VP2 から供給されます。誤って両者から同時に供給された場合、ショートを防ぐために VP1 側に 100 Ω の抵抗を挿入しています。データバスは 8 ビットモード時にはフルビット（DB7 ～ 0）が接続されるますが、4 ビットモード時には上位 4 ビット（DB7 ～ 4）だけ接続され、下位 4 ビット（DB3 ～ 0）は 0 に固定されます。

BZ（ブザー）信号（1 ビット）は VP1、VP2 のいずれかから供給されます。両者から同時に供給されてもショートしないように両者に適切な抵抗を挿入しています。BZ に交流電流だけ流すために適切なコンデンサを挿入しています。図では省略されていますが、各信号の H/L レベルをモニターするステータス LED を設置しています。

接続図の詳細は MT-E502 取扱説明書（http://www.sunhayato.co.jp/products/item_data/MT-E502_Manual_SG087044.pdf）をご参照ください。

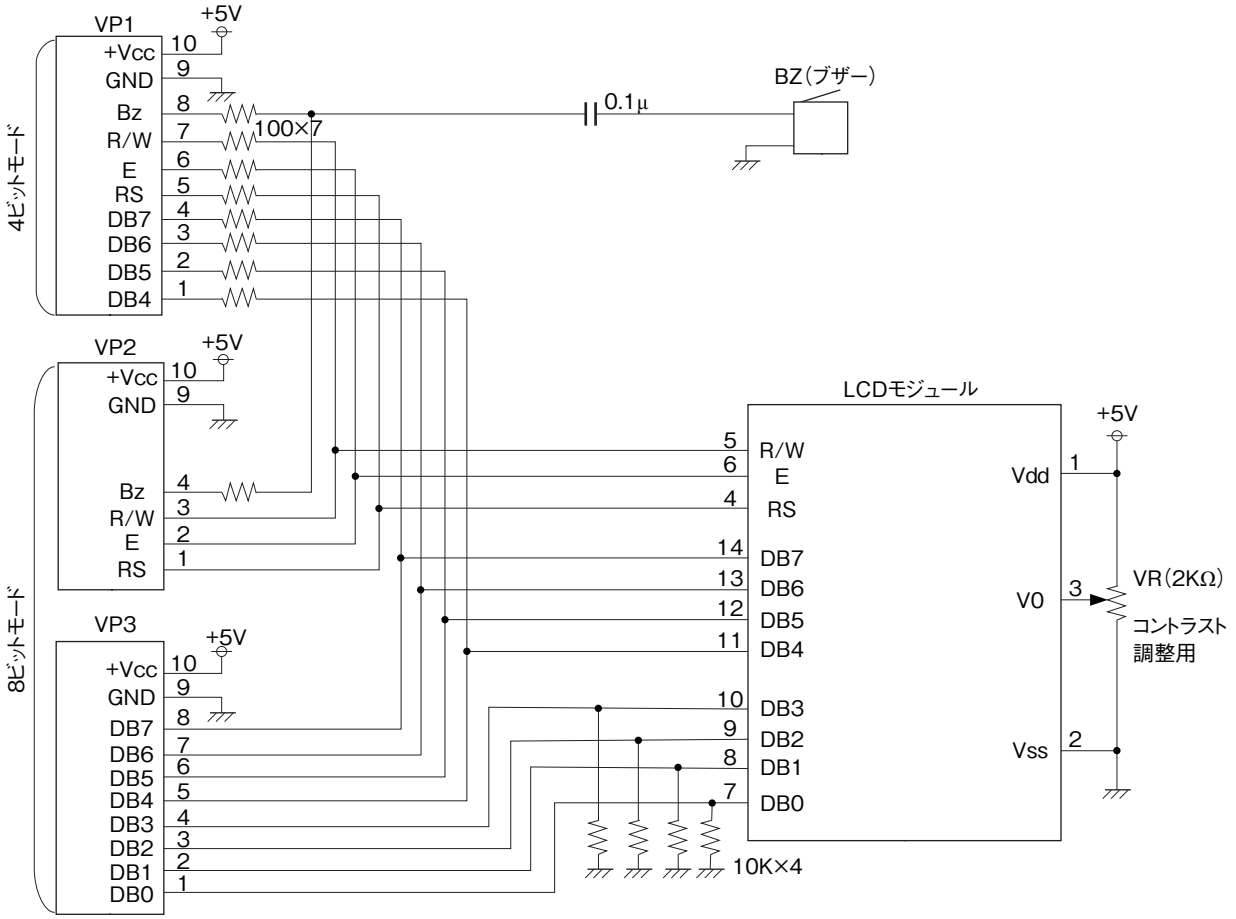


図 3-1　接続図の概要

4. 基本実習

まず 8 ビットモードで、種々のインストラクションを読み書きして、それに応じた制御を行います。インストラクションの書き込み時にはビジーチェック機能を付加し動作速度を向上させます。

これらの使い方を十分理解した後、そのプログラムを 4 ビットモード用に移植します。

最後に文字列や数値などを表示する関数を作成し、それらをまとめたライブラリーを作成します。

4.1 8 ビットモードでの制御

図 4-1 のように、付属の 10P ケーブルによりマイコンボードと LCD ボードを接続します。8 ビットモードでは VP2、VP3 の 2 つのポートを使います。前者のピンアサインは R/W 信号（b2）、E 信号（b1）、RS 信号（b0）となり、後者のそれは DB7 ～ 0 信号（b7 ～ 0）となります。これらを制御して LCD モジュールを制御します。

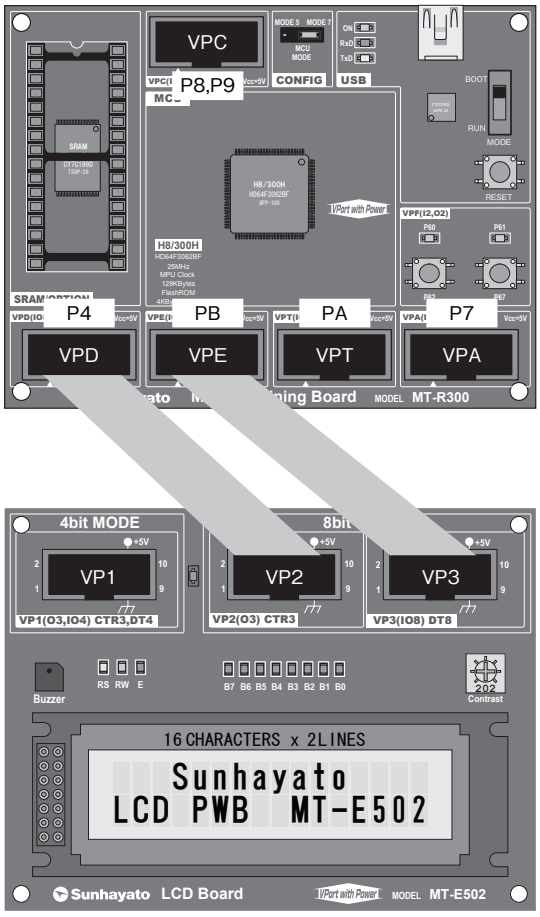


図 4-1　基板間の接続

ここでは "X:¥xxxx¥workspace" にワークスペース、プロジェクト（lcd_8b）を新規作成します。MT-R300 の実習と同様にハードウエアのセットアップ関数とメイン関数は自動生成しません。手動で "main.c" ファイル（空白）を新規作成し、プロジェクトに追加します。既に作成済みの "X:¥xxxx¥workspace¥@library" フォルダー内の "mt_r300.c" もプロジェクトに追加します。

種々のヘッダーファイルをインクルードするために次のとおり相対パスを指定します。

(1)［ビルド］、[H8S、H8/300 Standerd TOOLchain] を選択し、"H8S、H8/300 Standerd TOOLchain" ウインドウに切り替えます。

(2)［標準ライブラリー］タグ、［カテゴリ］で " ソース "、［オプション項目］で " インクルードファイルディレクトリ "（デフォルト値）を確認します。

(3)［追加］ボタンを押下して、［相対パス］で "Project directory" を選択し、［OK］を押下します。

(4) 同様に［追加］ボタンを押下して、［相対パス］で "Costom directory" を選択し、ディレクトリーに相対パスで "..¥..¥..¥@library" と入力し、[OK] を押下します。

実習 4.1.1　文字を表示する（DDRAM へのデータ書き込み）

ホームポジションから順次 'A'、'B'、'C' の文字を表示する。続いて 2 行目、4 列目から順次 '1'、'2'、'3' の文字を表示する。

［解説］

図 2-4 の各信号のタイミングチャートを参照してください。 次のように #difine 文を用いて、16 進数で各信号の状態（識別子）を定義します。

・R/W 信号（bit2）1: 読み出し（#difine RD 0x04）／ 0: 書き込み（#difine WT 0x00）
・E 信号（bit1）1: 動作許可（#difine EN 0x02）／ 0: 動作禁止（#difine DS 0x00）
・RS 信号（bit0）1: データレジスタ（#difine DR 0x01）／ 0; 命令レジスタ（#difine IR 0x00）

8 ビットモードで命令を書き込む手順は次のとおりです（図 2-4（a）参照）。

① VPD に（IR | WT | DS）を出力する
② VPE に命令（DB7 ～ 0）を出力する
③ VPD に（IR | WT | EN）を出力する
④動作許可時間 230ns 以上待つ（VPort 出力関数の実行時間が約 1μs かかるので、ウエイトはなくても問題なし）
⑤ VPD に（IR | WT | DS）を出力する
⑥サイクル時間 1200ns の不足時間だけ待つ

図 2-4 のタイミングチャートには示されていませんが、この後それぞれの命令実行時間（表 2-2 参照）待つ必要があります。例えば、表示クリア時は 1.53ms となります。従って、⑥の待ち時間は短いので命令実行時間に含めて考えます。

8 ビットモードでデータを書き込む手順は次のとおりです（図 2-4（b）参照）。

① VPD に（DR | WT | DS）を出力する
② VPE にデータ（DB7 ～ 0）を出力する
③ VPD に（DR | WT | EN）を出力する
④動作許可時間 230ns 以上待つ（適切なウエイト関数を作成します）
⑤ VPD に（DR | WT | DS）を出力する
⑥サイクル時間 1200ns の不足時間だけ待つ

同様にこの後 DDRAM へのデータ書き込み時間（43μs）待つ必要があります。従って、⑥の待ち時間は短いのでデータ書き込み時間に含めて考えます。

8 ビットモードで LCD を初期設定する手順は次のとおりです（図 2-6 参照）。

①電源投入後 45ms 以上待つ
②ファンクションセットで 8 ビットモード（0x30）に設定する（1 回目）
（DB5（1）、DB4（DL=1:8 ビット）、DB3（N=0:1 行）、DB2（F=0:5 × 7 ドット））
③ 4.1ms 以上待つ
④②と同じファンクションセットを繰り返す（2 回目）
⑤ 100μs 以上待つ
⑥②と同じファンクションセットを繰り返す（3 回目）
⑦ファンクションセットで、8 ビットモード、2 行表示（0x38）に設定する（DB5（1）、DB4（DL=1:8 ビット）、DB3（N=1:2 行）、DB2（F=*: 無効（5 × 7 ドット）））
⑧エントリーモードセットで、カーソル右移動、表示シフトしない（0x06）に設定する（DB2（1）、DB1（I/D=1: カーソル右移動）、DB0（S=0: 表示シフトしない））
⑨表示オン／オフで、表示オン、カーソルオフ、点滅オフ（0x0c）に設定する（DB3（1）、DB2（D=1: 表示オン）、DB1（C=0: カーソルオフ）、DB0（B=0: 点滅オフ））

"表示クリア"は設定ではないのでここでは初期設定に入れていません。従って、メイン関数で必要に応じ表示クリアして下さい。

[解答]

リスト 4.1.1 にプログラム例を示します。

リスト 4.1.1　文字を表示する（DDRAM へのデータ書き込み）

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>    // マイコンボード用ライブラリー
// VPD(b7 ～ 3: 未接続 ,b2:R/W(1: 読み出し /0: 書き込み ),b1:E(1: 許可 /0: 禁止 ,b0:RS(1: データ /0: 命令 ))
#define RD     0x04           // R/W=1 ( 読み出し )
#define WT     0x00           // R/W=0 ( 書き込み )
#define EN     0x02           // E=1   ( 動作許可 )
#define DS     0x00           // E=0   ( 動作禁止 )
#define DR     0x01           // RS=1  ( データ )
#define IR     0x00           // RS=0  ( 命令 )
void wait_ns(void)            // 実測値で約 710ns
{
}
void write_cmnd8_w(unsigned char cmnd)     // 8 ビットモードで命令を書き込む関数 ( ウエイト付 )
{
    out_VPD( IR | WT | DS );        // 動作禁止
    out_VPE(cmnd);                  // 命令 (8 ビット ) を書き込む
    out_VPD( IR | WT | EN );        // 動作許可 ( 別法 ) out_VPD( in_VPD() | EN);
    wait_ns();                      // 230ns 以上待つ
    out_VPD( IR | WT | DS );        // 動作禁止 ( 別法 ) out_VPD( in_VPD() & ~EN);
    wait_us_itu2(1530);                    // 命令実行時間待つ (最大 1.53ms (表示クリア時 ))
}
void write_data8_w(char dat)        // 8 ビットモードで文字を書込む関数 ( ウエイト付 )
{
    out_VPD( DR | WT | DS );        // 動作禁止
       wait_ms(500);                // 3 色 LED に注目 ( 確認後削除 )
    out_VPE(dat);                   // データ (8 ビット ) を書き込む
       wait_ms(500);                // 赤色 LED に注目 ( 確認後削除 )
    out_VPD( DR | WT | EN );        // 動作許可 ( 別法 ) out_VPD( in_VPD() | EN);
       wait_ms(500);                // 3 色 LED に注目 ( 確認後削除 )
    wait_ns();                      // 230ns 以上待つ
    out_VPD( DR | WT | DS );        // 動作禁止 ( 別法 ) out_VPD( in_VPD() & ~EN);
       wait_ms(500);                // 3 色 LED に注目 ( 確認後削除 )
    wait_us_itu2(43);               // 命令実行時間待つ (43us(DDRAM へのデータ書き込み ))
}
void initialize_lcd( void ) // LCD の初期化関数
{
    setup_VPD(0x07);          // VPD の初期化 (b2: 出力 (R/W), b1: 出力 (E), b0: 出力 (RS))
    setup_VPE(0xff);          // VPE の初期化 (b7 ～ 0: 出力 (DB7 ～ 0))
```

```
    wait_ms_itu2(45);        // LCD の準備時間（VDD が 4.5V に達してから 45ms 以上待つ）
    write_cmnd8_w(0x30);     // ファンクションセット（8 ビットモード）
                             // (DB5(1),DB4(DL=1:8 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    wait_us_itu2(4100);              // 4.1ms 以上待つ
    write_cmnd8_w(0x30);     // 同じファンクションセットを繰り返す（2 回目）
                             // (DB5(1),DB4(DL=1:8 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    wait_us_itu2(100);       // 100μs 以上待つ
    write_cmnd8_w(0x30);     // 同じファンクションセットを繰り返す（3 回目）
                             // (DB5(1),DB4(DL=1:8 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    write_cmnd8_w(0x38);     // ファンクションセット（8 ビットモード，2 行表示）
                    // (DB5(1),DB4(DL=1:8 ビット),DB3(N=1:2 行),DB2(F=*: 無効（5 × 7 ドット)))
    write_cmnd8_w(0x06);     // エントリーモードセット（カーソル右移動，表示シフトしない）
                             // (DB2(1),DB1(I/D=1: カーソル右移動),DB0(S=0: シフトしない))
    write_cmnd8_w(0x0c);     // 表示オン / オフ設定（表示オン，カーソルオフ，点滅オフ）
                    // (DB3(1),DB2(D=1: 表示オン),DB1(C=0: カーソルオフ),DB0(B=0: 点滅オフ))
}
void main(void){
    initialize_lcd();        // LCD の初期化
    while(1){
       write_cmnd8_w(0x01);                // 表示クリア（DB0(1))
       wait_ms(1000);
       write_cmnd8_w(0x02);                // リターンホーム（1 行目，1 列目）(DB1(1))
       write_data8_w('A');                 // 文字の表示（DDRAM へのデータ書き込み）
       write_data8_w('B');                 // 文字の表示（DDRAM へのデータ書き込み）
       write_data8_w('C');                 // 文字の表示（DDRAM へのデータ書き込み）
       wait_ms(1000);
       write_cmnd8_w(0x80 | 0x40+3);       // DDRAM アドレスセット（2 行目，4 列目）
                                           // (DB7(1),DB6 ～ 0(ADD))
       write_data8_w('1');                 // 文字の表示（DDRAM へのデータ書き込み）
       write_data8_w('2');                 // 文字の表示（DDRAM へのデータ書き込み）
       write_data8_w('3');                 // 文字の表示（DDRAM へのデータ書き込み）
       wait_ms(1000);
    }
}
```

書き込み時の動作許可時間 230ns は、次のウエイト関数を使います。余裕を見て 2 倍以上とします。

```
    void wait_ns(void)               // 実測値で約 710ns
    {
    }
```

R/W、E、RS の各信号（H/L）の変化を LED でモニタするために次のウエイトを挿入しました。それぞれの信号の変化を確認して下さい。

```
    wait_ms(500);  // LED に注目
```

実習 4.1.2　ビジーチェックプログラムを作成する

文字を表示しながら BF/AC（ビジーフラグ / アドレスカウンタ）を読み出し、評価用 LED で確認をします。その後ビジーチェックプログラムを作成します。評価用 LED には標準 I/O ボード MT-E501 の中央 LED(VP2) を用いるので、それを MT-R300 の VPT に接続します。

[解説]

LCD から 8 ビットモードで状態（BF/AC）を読み出す手順は次のとおりです（図 4（c）参照）。

① VPE を一時的に（入力モード）に設定する
② VPD に（IR | RD | DS）を出力する
③ VPD に（IR | RD | EN）を出力する
④ 動作許可時間 230ns 以上待つ
⑤ VPE から（BF/AC）を入力する
⑥ VPD に（IR | RD | DS）を出力する
⑦ VPE を本来の（出力モード）に戻す
⑧ return (BF/AC)

LCD の初期設定で、3 回のファンクションセット（8 ビットモード設定）時はビジー信号が無効のため、それぞれ指定の時間（サイクル時間の不足時間も含む）を待ちます。それ以降はビジー信号が有効になるため、インストラクションの書き込み前にビジーチェックを行います。これによりビジー状態が終了するまで待たされるため、サイクル時間の不足時間や命令実行時間などを待つ処理は不要になります。

[解答]

リスト 4.1.2 にプログラム例を示します。

リスト 4.1.2　文字を表示し、状態（BF/AC）を読み出し LED で確認する。その後ビジーチェックプログラムを作成する。

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>    // マイコンボード用ライブラリー
// VPD(b7 ～ 3: 未接続 ,b2:R/W(1: 読み出し /0: 書き込み ),b1:E(1: 許可 /0: 禁止 ,b0:RS(1: データ /0: 命令 ))
#define RD     0x04           // R/W=1 ( 読み出し )
#define WT     0x00           // R/W=0 ( 書き込み )
#define EN     0x02           // E=1   ( 動作許可 )
#define DS     0x00           // E=0   ( 動作禁止 )
#define DR     0x01           // RS=1  ( データ )
#define IR     0x00           // RS=0  ( 命令 )
void wait_ns(void)            // 実測値で約 710ns
{
}
unsigned char read_cmnd8( void )    // BF( ビジーフラグ ),AC( アドレスカウンタ ) を読み出す関数
{
    unsigned char cmnd;
    setup_VPE(0x00);                // VPE を一時的に入力モードに設定
    out_VPD( IR | RD | DS );        // 動作禁止
    out_VPD( IR | RD | EN );        // 動作許可
    wait_ns();                      // 230ns 以上待つ
```

```
    cmnd = in_VPE();                // BF/AC を読み出す
    out_VPD( IR | RD | DS );        // 動作禁止
    setup_VPE(0xff);                // VPE を本来の出力モードに再設定
    return cmnd;
}
void write_cmnd8(unsigned char cmnd)       // 命令を書き込む関数
{
    out_VPD( IR | WT | DS );        // 動作禁止
    out_VPE(cmnd);                  // 命令（8 ビット ) を書き込む
    out_VPD( IR | WT | EN );        // 動作許可
    wait_ns();                      // 230ns 以上待つ
    out_VPD( IR | WT | DS );        // 動作禁止
}
void write_cmnd8_bc(unsigned char cmnd)    // 命令を書込む関数（ビジーチェック付）
{
    while(read_cmnd8() & 0x80);     // ビジー（BF=1(0x80)) のとき繰り返す ( 待つ )
    out_VPD( IR | WT | DS );        // 動作禁止
    out_VPE(cmnd);                  // 命令（8 ビット ) を書き込む
    out_VPD( IR | WT | EN );        // 動作許可
    wait_ns();                      // 230ns 以上待つ
    out_VPD( IR | WT | DS );        // 動作禁止
}
void write_data8_bc(char dat)       // データを書き込む関数（ビジーチェック付）
{
//  次の 4 行は，LED で BF/AC の状態を確認するものです（通常はこの 4 行が不要です )
    out_VPT( read_cmnd8() );        // LED で BF(b7),AC(b6 ～ 0) を確認する（一瞬 BF=1 になる )
    wait_ms(1000);
    out_VPT( read_cmnd8() );        // LED で BF(b7),AC(b6 ～ 0) を確認する（BF=0 で，AC は +1)
    wait_ms(1000);
    while(read_cmnd8() & 0x80);     // ビジー（BF=1(0x80)) のとき繰り返す ( 待つ )
    out_VPD( DR | WT | DS );        // 動作禁止
    out_VPE(dat);                   // データ（8 ビット ) を書き込む
    out_VPD( DR | WT | EN );        // 動作許可
    wait_ns();                      // 230ns 以上待つ
    out_VPD( DR | WT | DS );        // 動作禁止
}
void initialize_lcd( void ) // LCD の初期化関数
{
    setup_VPD(0x07);          // VPD の初期化（b2: 出力 (R/W), b1: 出力 (E), b0: 出力 (RS))
    setup_VPE(0xff);          // VPE の初期化（b7 ～ 0: 出力 (DB7 ～ 0))
    wait_ms_itu2(45);         // LCD の準備時間 (VDD が 4.5V に達してから 45ms 以上待つ )
    write_cmnd8(0x30);        // ファンクションセット (8 ビットモード )
    wait_us_itu2(4100);             // 4.1ms 以上待つ
    write_cmnd8(0x30);        // 同じファンクションセットを繰り返す (2 回目 )
    wait_us_itu2(100);        // 100μs 以上待つ
```

```
    write_cmnd8(0x30);       // 同じファンクションセットを繰り返す（3 回目 ）
    write_cmnd8_bc(0x38);    // ファンクションセット（8 ビットモード，2 行表示 ）
    write_cmnd8_bc(0x06);    // エントリーモードセット（ カーソル右移動 , 表示シフトしない ）
    write_cmnd8_bc(0x0c);    // 表示オン / オフ設定（ 表示オン , カーソルオフ , 点滅オフ ）
}
void main(void){
    initialize_lcd();                    // LCD の初期化
    setup_VPT(0xff);                     // VPT の初期化（b7 ～ 0: 出力 (LED)）
    while(1){
       write_cmnd8_bc(0x01);             // 表示クリア (DB0(1))
       wait_ms(1000);
       write_cmnd8_bc(0x02);             // リターンホーム（1 行目 ,1 列目 ）
       write_data8_bc('A');              // 文字の表示
       write_data8_bc('B');              // 文字の表示
       write_data8_bc('C');              // 文字の表示
       write_data8_bc('D');              // 文字の表示
       write_data8_bc('E');              // 文字の表示
       write_cmnd8_bc(0x80 | 0x40+3);    // DDRAM アドレスセット（2 行目 ,4 列目 ）
       write_data8_bc('1');              // 文字の表示
       write_data8_bc('2');              // 文字の表示
       write_data8_bc('3');              // 文字の表示
       write_data8_bc('4');              // 文字の表示
       write_data8_bc('5');              // 文字の表示
       wait_ms(1000);
    }
}
```

"out_VPT(read_cmnd8());　　// LED で BF(b7),AC(b6 ～ 0) を確認する " の 2 行分は、MT-E501 の LED で BF/AC を確認するためのプログラムです。ビジーフラグ BF（b7）は一瞬だけ 1 になり、すぐに 0 に変わるため、その変化を確認できません。しかし、2 行のプログラムの間に大きなウエイトを挿入すると、最初に BF=1 を確認でき、次に BF=0 を確認できます。アドレスカウンタ AC（b6 ～ 0）は BF=0 に変化したときに +1 され、次に表示する場所（アドレス）を表示します。

ビジーチェックは "while(read_cmnd8() & 0x80);" で行います。LCD モジュールの BF（b7）/AC（b6 ～ 0）を読み出し、ビジー（0x80）の間 while 文を繰り返し、レディー（0x00）になると while 文を抜けて次の書き込み動作に進みます。

実習 4.1.3　LCD の 1 行目に文字を表示し、それを読み出し（演算を加え）、2 行目に表示する

[解説]

LCD から 8 ビットモードでデータを読み出す手順は次のとおりです（図 2-4（d）参照）。

① VPE を一時的に（入力モード）に設定する
② VPD に（DR | RD | DS）を出力する
③ VPD に（DR | RD | EN）を出力する
④動作許可時間 230ns 以上待つ

⑤ VPE からデータを入力する
⑥ VPD に（DR | RD | DS）を出力する
⑦ VPE を本来の（出力モード）に戻す
⑧ return（データ）

データ書き込み時にビジーチェックしましたが、同様にデータ読み込み時にもビジーチェックを行います。

[解答]

リスト 4.1.3 にプログラム例を示します。

リスト 4.1.3　LCD の 1 行目に文字を表示し、それを読み出し（演算を加え）、2 行目に表示する

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>    // マイコンボード用ライブラリー
// VPD(b7 ～ 3: 未接続 ,b2:R/W(1: 読み出し /0: 書き込み ),b1:E(1: 許可 /0: 禁止 ,b0:RS(1: データ /0: 命令 ))
#define RD     0x04           // R/W=1 ( 読み出し )
#define WT     0x00           // R/W=0 ( 書き込み )
#define EN     0x02           // E=1   ( 動作許可 )
#define DS     0x00           // E=0   ( 動作禁止 )
#define DR     0x01           // RS=1  ( データ )
#define IR     0x00           // RS=0  ( 命令 )
void wait_ns(void)            // 実測値で約 710ns
{
}
unsigned char read_cmnd8( void )    // BF( ビジーフラグ ),AC( アドレスカウンタ ) を読み出す関数
{
    unsigned char cmnd;
    setup_VPE(0x00);                // VPE を一時的に入力に設定
    out_VPD( IR | RD | DS );        // 動作禁止
    out_VPD( IR | RD | EN );        // 動作許可
    wait_ns();                      // 230ns 以上待つ
    cmnd = in_VPE();                // BF/AC を読み出す
    out_VPD( IR | RD | DS );        // 動作禁止
    setup_VPE(0xff);                // VPE を本来の出力に再設定
    return cmnd;
}
unsigned char read_data8( void )    // DDRAM からデータを読み出す関数
{
    unsigned char d;
    while(read_cmnd8() & 0x80);     // ビジー (BF=1(0x80)) のとき繰り返す ( 待つ )
    setup_VPE(0x00);                // VPE を一時的に入力に設定
    out_VPD( DR | RD | DS );        // 動作禁止
    out_VPD( DR | RD | EN );        // 動作許可
    wait_ns();                      // 230ns 以上待つ
    d = in_VPE();                   // データを読み出す
    out_VPD( DR | RD | DS );        // 動作禁止
```

```
    setup_VPE(0xff);                // VPE を本来の出力に再設定
    return d;
}
void write_cmnd8(unsigned char cmnd)       // 命令を書き込む関数
{
    out_VPD( IR | WT | DS );        // 動作禁止
    out_VPE(cmnd);                  // 命令（8 ビット）を書き込む
    out_VPD( IR | WT | EN );        // 動作許可
    wait_ns();                      // 230ns 以上待つ
    out_VPD( IR | WT | DS );        // 動作禁止
}
void write_cmnd8_bc(unsigned char cmnd)    // 命令を書込む関数（ビジーチェック付）
{
    while(read_cmnd8() & 0x80);     // ビジー（BF=1(0x80)）のとき繰り返す（待つ）
    out_VPD( IR | WT | DS );        // 動作禁止
    out_VPE(cmnd);                  // 命令（8 ビット）を書き込む
    out_VPD( IR | WT | EN );        // 動作許可
    wait_ns();                      // 230ns 以上待つ
    out_VPD( IR | WT | DS );        // 動作禁止
}
void write_data8_bc(char dat)       // データを書き込む関数（ビジーチェック付）
{
    while(read_cmnd8() & 0x80);     // ビジー（BF=1(0x80)）のとき繰り返す（待つ）
    out_VPD( DR | WT | DS );        // 動作禁止
    out_VPE(dat);                   // データ（8 ビット）を書き込む
    out_VPD( DR | WT | EN );        // 動作許可
    wait_ns();                      // 230ns 以上待つ
    out_VPD( DR | WT | DS );        // 動作禁止
}
void initialize_lcd( void )         // LCD の初期化関数
{
    setup_VPD(0x07);          // VPD の初期化（b2: 出力（R/W），b1: 出力（E），b0: 出力（RS））
    setup_VPE(0xff);          // VPE の初期化（b7 ～ 0: 出力（DB7 ～ 0））
    wait_ms_itu2(45);         // LCD の準備時間（VDD が 4.5V に達してから 45ms 以上待つ）
    write_cmnd8(0x30);        // ファンクションセット（8 ビットモード）
    wait_us_itu2(4100);       // 4.1ms 以上待つ
    write_cmnd8(0x30);        // 同じファンクションセットを繰り返す（2 回目）
    wait_us_itu2(100);        // 100μs 以上待つ
    write_cmnd8(0x30);        // 同じファンクションセットを繰り返す（3 回目）
    write_cmnd8_bc(0x38);     // ファンクションセット（8 ビットモード，2 行表示）
    write_cmnd8_bc(0x06);     // エントリーモードセット（カーソル右移動，表示シフトしない）
    write_cmnd8_bc(0x0c);     // 表示オン / オフ設定（表示オン，カーソルオフ，点滅オフ）
}
void main(void){
    int i;
```

```
    unsigned char d;
    initialize_lcd();               // LCD の初期化
    while(1){
       write_cmnd8_bc(0x0c);        // 表示オン／オフで，表示オン，カーソルオフ，点滅オフに設定
       write_cmnd8_bc(0x01);        // 表示クリア (DB0(1))
       wait_ms(1000);
       write_cmnd8_bc(0x02);        // リターンホーム（1 行目 ,1 列目 ）
       write_data8_bc('A');         // 文字の表示
       write_data8_bc('B');
       write_data8_bc('C');
       write_data8_bc('D');
       write_data8_bc('E');
       write_data8_bc('F');
       wait_ms(1000);
       // *** 文字を読み出し，それを（演算して）下の行に表示する ***
       write_cmnd8_bc(0x0f);        // 表示オン / オフで，表示オン，カーソルオン，点滅オンに設定
       for(i = 0; i < 6; i++){
              write_cmnd8_bc(0x80 | 0x00+i);      // DDRAM アドレスセット（1 行目 ,i 列目 ）
              d = read_data8();                   // 文字を読み出す
              write_cmnd8_bc(0x80 | 0x40+i);      // DDRAM アドレスセット（2 行目 ,i 列目 ）
              write_data8_bc(d);                  // 文字をそのまま表示する ("ABCDEF")
//            write_data8_bc(d + 0x20);           // 文字を演算して表示する ("abcdef")
//            write_data8_bc(d - 0x10);           // 文字を演算して表示する ("123456")
              wait_ms(200);
       }
       wait_ms(1000);
    }
}
```

4.2　4 ビットモードでの制御

図 4-2 のように、付属の 10P ケーブルによりマイコンボードと LCD ボードを接続します。4 ビットモードでは、VP1 ポートのみを使います。ピンアサインは、R/W 信号（bit6）、E 信号（bit5）、RS 信号（bit4）、DB7 ～ 4 信号（bit3 ～ 0）となります。DB3 ～ 0 信号はプルダウンに固定されています。データ信号は上位 4 ビット、下位 4 ビットの順に 2 回に分けて読み書きします。

ここでは、"X:¥xxxx¥workspace" にワークスペース、プロジェクト（lcd_4b）を新規作成し、実習します。4.1 と同様に種々の設定を行います。

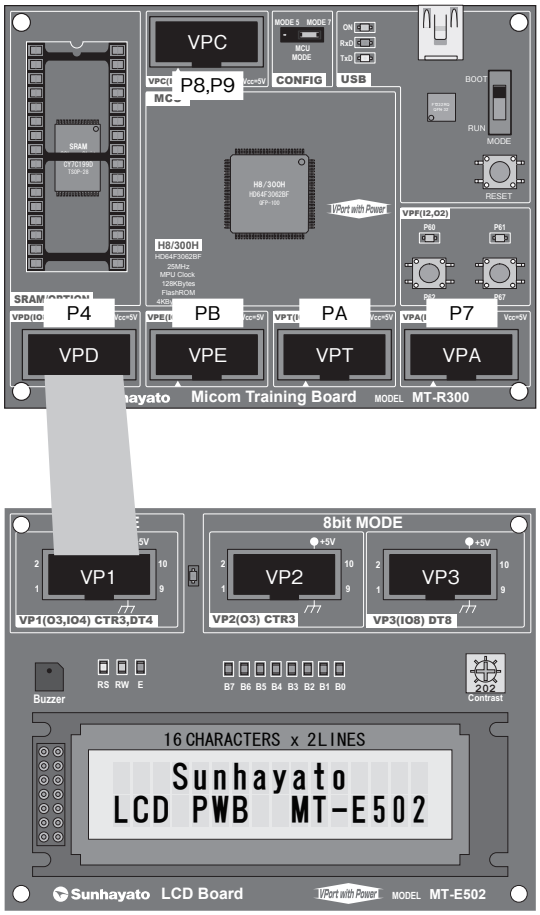


図 4-2　基板間の接続

実習 4.2.1　文字を表示する（DDRAM へのデータ書き込み）

ホームポジションから 'A'、'B'、'C' の文字を表示します。続いて 2 行目、4 列目から '1'、'2'、'3' の文字を表示します。

[解説]

図 2-4 の信号線のタイミングチャートを参照してください。次のように #difine 文を用いて 16 進数で各信号の状態（識別子）を定義します。8 ビットモードの時とビットの対応が変わりますので、注意してください。

・R/W 信号（bit6）1: 読み出し（#difine RD 0x40）／ 0: 書き込み（#difine WT 0x00）
・E 信号（bit5）1: 動作許可（#difine EN 0x20）／ 0: 動作禁止（#difine DS 0x00）
・RS 信号（bit4）1: データレジスタ（#difine DR 0x10）／ 0: 命令レジスタ（#difine IR 0x00）

4 ビットモードで命令を書き込む場合は、上位 4 ビット、下位ビット（左に 4 ビットシフト後）の順に 2 回に分けて書き込みます。

4 ビットモードで LCD を初期設定する手順は次のとおりです。

①電源投入後 45ms 以上待つ
②ファンクションセットで 8 ビットモード（0x30）に設定する（1 回目）
（DB5（1）、DB4（DL=1:8 ビット）、DB3（N=0:1 行）、DB2（F=0:5 × 7 ドット））
③ 4.1ms 以上待つ
④②と同じファンクションセットを繰り返す（2 回目）
⑤ 100μs 以上待つ
⑥②と同じファンクションセットを繰り返す（3 回目）
⑦ 39μs 以上待つ（ここからビジーチェックが可能ですが、ここではウエイトで対応します）
⑧ファンクションセットで 4 ビットモード（0x20）に変更する（DB5（1）、DB4（DL=0:4 ビット）、DB3（N=0:1 行）、DB2（F=0:5 × 7 ドット））（これ以降は 4 ビットモードで動作します）
⑨ファンクションセットで、4 ビットモード、2 行表示（0x28）に設定する（DB5（1）、DB4（DL=0:4 ビット）、DB3（N=1:2 行）、DB2（F=*: 無効（5 × 7 ドット）））。実際には（0x20）、（0x80）の順に書き込みます（39μs

以上のウエイトまたはビジーチェック）

⑩エントリーモードセットで、表示シフトしない (0x06) に設定する（DB2（1）、DB1（I/D=1: カーソル右移動）、DB0（S=0: シフトしない））。実際には（0x00）、(0x60）の順に書き込みます（39μs 以上のウエイトまたはビジーチェック）

⑪表示オン／オフで、表示をオン、カーソルをオフ、点滅をオフ（0x0c）に設定する（DB3（1）、DB2（D=1: 表示オン）、DB1（C=0: カーソルオフ）、DB0（B=0: 点滅オフ））。実際には（0x00）、(0xc0）の順に設定します（39μs 以上のウエイトまたはビジーチェック）

[解答]

リスト 4.2.1 にプログラム例を示します。

リスト 4.2.1　文字を表示する（DDRAM へのデータ書き込み）

```
/** main.c ********************************************************************************/
#include <mt_r300.h> // マイコンボード用ライブラリー
// VP1(b7: 未接続 ,b6:R/W(1: 読み出し /0: 書き込み ),b5:E(1: 許可 /0: 禁止 ,b4:RS(1: データ /0: 命令 ),b3 ～ 0
(DB7 ～ 4))
#define RD      0x40                        // R/W=1 ( 読み出し )
#define WT      0x00                        // R/W=0 ( 書き込み )
#define EN      0x20                        // E=1   ( 動作許可 )
#define DS      0x00                        // E=0   ( 動作禁止 )
#define DR      0x10                        // RS=1  ( データ )
#define IR      0x00                        // RS=0  ( 命令 )

void wait_ns(void)                          // 実測値で約 710ns
{
}

void write_cmnd8_w(unsigned char cmnd)      // 命令を書き込む関数（8 ビットモードで 1 回で、ウエイトなし）
{
    out_VPD( IR | WT | DS );                // 動作禁止
    out_VPD( IR | WT | EN | (cmnd >> 4) ); // 動作許可，上位 4 ビットを下位 4 ビットにシフトし書き込む
    wait_ns();                              // 230ns 以上待つ
    out_VPD( IR | WT | DS | (cmnd >> 4) ); // 動作禁止
    wait_us_itu2(39);                       // 命令実行時間待つ（39μs（ファンクションセット））
}

void write_cmnd4_w(unsigned char cmnd)      // 命令を書き込む（4 ビットモードで）
{
    out_VPD( IR | WT | DS );                // 動作禁止
    out_VPD( IR | WT | EN | (cmnd >> 4) ); // 動作許可，上位 4 ビットを書き込む（シフト付）
//  wait_ns();                              // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( IR | WT | DS | (cmnd >> 4) ); // 動作禁止
//  wait_ns();                              // サイクルタイム（1200ns 以上）を確保（可能なら省略）
                                            // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( IR | WT | DS );                // 動作禁止，直前の命令（命令付）に吸収する
```

```
    out_VPD( IR | WT | EN | (cmnd & 0x0f) );      // 動作許可，下位 4 ビットを書き込む
//  wait_ns();                                    // イネーブル信号幅（230ns 以上 ）を確保（ 可能なら省略 ）
    out_VPD( IR | WT | DS | (cmnd & 0x0f) );      // 動作禁止
    wait_us_itu2(1530);                           // 命令実行時間待つ（1.53ms（表示クリアなど ））
}

void write_data4_w(char data)       // 4 ビットモードで 2 回に分けてデータを書き込む関数（ ウエイト付 ）
{
    out_VPD( DR | WT | DS );                     // 動作禁止
    out_VPD( DR | WT | EN | (data >> 4) ); // 動作許可，上位 4 ビットを書き込む（シフト付 ）
//  wait_ns();                                    // イネーブル信号幅（230ns 以上 ）を確保（ 可能なら省略 ）
    out_VPD( DR | WT | DS | (data >> 4) ); // 動作禁止
//  wait_ns();                                    // サイクルタイム（1200ns 以上 ）を確保（ 可能なら省略 ）
                                                  // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( DR | WT | DS );                     // 動作禁止，直前の命令（ データ付 ）に吸収する
    out_VPD( DR | WT | EN | (data & 0x0f) );      // 動作許可，下位 4 ビットを書き込む
//  wait_ns();                                    // イネーブル信号幅（230ns 以上 ）を確保（ 可能なら省略 ）
    out_VPD( DR | WT | DS | (data & 0x0f) );      // 動作禁止
    wait_us_itu2(43);                             // 命令実行時間待つ（43μs(DDRAM へのデータ書き込み ））
}

void initialize_lcd( void )                   // LCD の初期化関数
{
    setup_VPD(0xff);         // VPD の初期化（b7:(BZ),b6:(R/W),b5:(E),(b4:RS),b3 ～ 0:(DB7 ～ 4)）
    wait_ms_itu2(45);                             // LCD の準備時間（VDD が 4.5V に達してから 45ms 以上待つ ）
    write_cmnd8_w(0x30);                          // まずファンクションセットで，8 ビットモードに設定する
                      // （DB5(1),DB4(DL=1:8 ビット ),DB3(N=0:1 行 ),DB2(F=0:5 × 7 ドット ））
    wait_us_itu2(4100);                           // 4.1ms 以上待つ
    write_cmnd8_w(0x30);                          // 同じファンクションセットを繰り返す（2 回目 ）
                      // （DB5(1),DB4(DL=1:8 ビット ),DB3(N=0:1 行 ),DB2(F=0:5 × 7 ドット ））
    wait_us_itu2(100);                            // 100 μ s 以上待つ
    write_cmnd8_w(0x30);                          // 同じファンクションセットを繰り返す（3 回目 ）
                      // （DB5(1),DB4(DL=1:8 ビット ),DB3(N=0:1 行 ),DB2(F=0:5 × 7 ドット ））
    write_cmnd8_w(0x20);                          // ファンクションセットで，4 ビットモードに設定する
                      //  （DB5(1),DB4(DL=0:4 ビット ),DB3(N=0:1 行 ),DB2(F=0:5 × 7 ドット ））
    write_cmnd4_w(0x28);                          // ファンクションセットで，4 ビットモード，2 行表示に設定する
                      //  （DB5(1),DB4(DL=0:4 ビット ),DB3(N=1:2 行 ),DB2(F=*: 無効（5 × 7 ドット ）））
    write_cmnd4_w(0x06);                          // エントリーモードセットで，表示をシフトしないに設定する
                      // （DB2(1),DB1(I/D=*: 無効 ),DB0(S=0: 表示シフトしない ））
    write_cmnd4_w(0x0c);            // 表示オン／オフで，表示をオン，カーソルをオフ，点滅をオフにする
                      // （DB3(1),DB2(D=1: 表示オン ),DB1(C=0: カーソルオフ ),DB0(B=0: 点滅オフ ））
}

void main(void){
    initialize_lcd();                             // LCD の初期化
```

```
    while(1){
        write_cmnd4_w(0x01);                // 表示クリア (DB0(1))
        wait_ms(500);
        write_cmnd4_w(0x02);                // リターンホーム (DB1(1))
        write_data4_w('A');                 // 文字の表示 (DDRAM へのデータ書き込み )
        write_data4_w('B');
        write_data4_w('C');
        wait_ms(1000);
        write_cmnd4_w(0x80 | 0x40+3);// DDRAM アドレスセット (2 行目 ,4 列目 ) (DB7(1),DB6 ～ 0(ADD))
        write_data4_w('1');                 // 文字の表示 (DDRAM へのデータ書き込み )
        write_data4_w('2');
        write_data4_w('3');
        wait_ms(1000);
    }
}
```

実習 4.2.2 ビジーチェックプログラムを作成する

文字を表示しながら状態（BF/AC）を読み出し、評価用 LED で確認をします。その後ビジーチェックプログラムを作成します。評価用 LED には標準 I/O ボード MT-E501 の中央 LED(VP2) を用いるので、それを MT-R300 の VPT に接続します。

[解説]

4 ビットモードで命令の半分を読み出す手順は次のとおりです。

① VPE を一時的に（入力モード）に設定する
② VPD に（IR | RD | DS）を出力する
③ VPD に（IR | RD | EN）を出力する
④ VPE から（BF/AC）の半分を入力する
⑤ VPD に（IR | RD | DS）を出力する
⑥ VPE を（出力モード）に設定する
⑦ return（BF/AC）の半分

これを上位 4 ビット（左に 4 ビットシフト後)、下位 4 ビットの順に読み込み 8 ビットに合成します。

[解答]

リスト 4.2.2 にプログラム例を示します。

リスト 4.2.2　文字を表示し、BF/AC を読み出し評価用 LED で確認をする。その後ビジーチェックプログラムを作成する。

```
/** main.c ********************************************************************************/
#include <mt_r300.h> // マイコンボード用ライブラリー
// VP1(b7: 未接続 ,b6:R/W(1: 読み出し /0: 書き込み ),b5:E(1: 許可 /0: 禁止 ,b4:RS(1: データ /0: 命令 ),b3 ～ 0
(DB7 ～ 4))
#define RD      0x40                        // R/W=1 ( 読み出し )
#define WT      0x00                        // R/W=0 ( 書き込み )
#define EN      0x20                        // E=1   ( 動作許可 )
```

```
#define DS      0x00                        // E=0   ( 動作禁止 )
#define DR      0x10                        // RS=1  ( データ )
#define IR      0x00                        // RS=0  ( 命令 )

void wait_ns(void)                          // 動作許可時間（実測値で約 710ns）
{
}

unsigned char read_cmnd4( void )            // 命令 ( ステータス ) を読み出す（4 ビットモードで）
{
    unsigned char cmnd;
    setup_VPD(0xf0);                        // VPD の下位 4 ビットを一時的に入力モードに設定
    out_VPD( IR | RD | DS );                // 動作禁止
    out_VPD( IR | RD | EN );                // 動作許可
//  wait_ns();                              // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    cmnd = in_VPD() << 4;                   // 上位 4 ビットを読み出す ( シフト付 )
    out_VPD( IR | RD | DS );                // 動作禁止
//  wait_ns();                              // サイクルタイム (1200ns 以上 ) を確保 ( 可能なら省略 )
                                            // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( IR | RD | DS );                // 動作禁止，直前の命令に吸収する
    out_VPD( IR | RD | EN );                // 動作許可
//  wait_ns();                              // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    cmnd |= in_VPD() & 0x0f;                    // 下位 4 ビットを読み出し、上位 4 ビットと合成する
    out_VPD( IR | RD | DS );                    // 動作禁止
    setup_VPD(0xff);                            // VPD の下位 4 ビットを本来の出力モードに再設定
    return cmnd;
}

void write_cmnd8(unsigned char cmnd)            // 命令を書き込む関数（8 ビットモードで, ウエイトなし）
{
    out_VPD( IR | WT | DS );                    // 動作禁止
    out_VPD( IR | WT | EN | (cmnd >> 4) ); // 動作許可，上位 4 ビットを下位 4 ビットにシフトし書き込む
    wait_ns();                                  // 230ns 以上待つ
    out_VPD( IR | WT | DS | (cmnd >> 4) );      // 動作禁止
}

void write_cmnd4_bc(unsigned char cmnd)         // 命令を書き込む (4 ビットモードで, ビジーチェック付 )
{
    while(read_cmnd4() & 0x80);                 // ビジー (BF=1(0x80)) のとき繰り返す ( 待つ )
    out_VPD( IR | WT | DS );                    // 動作禁止
    out_VPD( IR | WT | EN | (cmnd >> 4) );      // 動作許可，上位 4 ビットを書き込む（シフト付 )
//  wait_ns();                              // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    out_VPD( IR | WT | DS | (cmnd >> 4) );      // 動作禁止
//  wait_ns();                                  // サイクルタイム (1200ns 以上 ) を確保 ( 可能なら省略 )
                                            // VPort のアクセスタイムが約 1000ns のためウエイトを省略
```

```
//  out_VPD( IR | WT | DS );                        // 動作禁止，直前の命令（命令付）に吸収する
    out_VPD( IR | WT | EN | (cmnd & 0x0f) );        // 動作許可，下位 4 ビットを書き込む
//  wait_ns();                                      // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( IR | WT | DS | (cmnd & 0x0f) );        // 動作禁止
}

void write_data4_bc(char data)                      // 文字を書き込む（4 ビットモードで，ビジーチェック付）
{
    // 次の 4 行は，MT-E501 の LED で BF/AC の状態をモニタするものです（通常はこの 4 行が不要です）
       out_VPT( read_cmnd4() );                     // LED で BF(b7)/AC(b6 ～ 0) を確認する．（一瞬 BF=1 になる）
       wait_ms(1000);
       out_VPT( read_cmnd4() );     // LED で BF(b7)/AC(b6 ～ 0) を確認する．(BF=0 になり，AC が +1 される）
       wait_ms(1000);
    while(read_cmnd4() & 0x80);                     // ビジー（BF=1(0x80)）のとき繰り返す（待つ）
    out_VPD( DR | WT | DS );                        // 動作禁止
    out_VPD( DR | WT | EN | (data >> 4) );          // 動作許可，上位 4 ビットを書き込む（シフト付）
//  wait_ns();                                      // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( DR | WT | DS | (data >> 4) );          // 動作禁止
//  wait_ns();                                      // サイクルタイム（1200ns 以上）を確保（可能なら省略）
                                                    // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( DR | WT | DS );                        // 動作禁止，直前の命令（データ付）に吸収する
    out_VPD( DR | WT | EN | (data & 0x0f) );        // 動作許可，下位 4 ビットを書き込む
//  wait_ns();                                      // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( DR | WT | DS | (data & 0x0f) );        // 動作禁止
}

void initialize_lcd( void ) // LCD の初期化関数
{
    setup_VPD(0xff);          // VPD の初期化（b7:(BZ),b6:(R/W),b5:(E),(b4:RS),b3 ～ 0:(DB7 ～ 4)）
    wait_ms_itu2(45);         // LCD の準備時間（VDD が 4.5V に達してから 45ms 以上待つ）
    write_cmnd8(0x30);        // まずファンクションセットで，8 ビットモードに設定する
                              // (DB5(1),DB4(DL=1:8 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    wait_us_itu2(4100);       // 4.1ms 以上待つ
    write_cmnd8(0x30);        // 同じファンクションセットを繰り返す（2 回目）
                              // (DB5(1),DB4(DL=1:8 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    wait_us_itu2(100);        // 100μs 以上待つ
    write_cmnd8(0x30);        // 同じファンクションセットを繰り返す（3 回目）
                              // (DB5(1),DB4(DL=1:8 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    wait_us_itu2(39);         // 39μs（ファンクションセット命令実行時間）以上待つ
    write_cmnd8(0x20);        // ファンクションセットで，4 ビットモードに設定する
                              //  (DB5(1),DB4(DL=0:4 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    write_cmnd4_bc(0x28);     // ファンクションセットで，4 ビットモード，2 行表示に設定する
                      //  (DB5(1),DB4(DL=0:4 ビット),DB3(N=1:2 行),DB2(F=*: 無効（5 × 7 ドット)))
    write_cmnd4_bc(0x06);     // エントリーモードセットで，表示をシフトしないに設定する
                              // (DB2(1),DB1(I/D=*: 無効),DB0(S=0: 表示シフトしない))
```

```
    write_cmnd4_bc(0x0c);    // 表示オン／オフで，表示をオン，カーソルをオフ，点滅をオフにする
                             // (DB3(1),DB2(D=1:表示オン),DB1(C=0:カーソルオフ),DB0(B=0:点滅オフ))
}

void main(void){
    setup_VPT(0xff);                        // VPT の初期化（b7 ～ 0: 出力 (LED))
    initialize_lcd();                       // LCD の初期化
    while(1){
       write_cmnd4_bc(0x01);                // 表示クリア (DB0(1))
       wait_ms(1000);
       write_cmnd4_bc(0x02);                // リターンホーム（DB1(1))
       write_data4_bc('A');                 // 文字の表示
       write_data4_bc('B');
       write_data4_bc('C');
       write_data4_bc('D');
       write_data4_bc('E');

       write_cmnd4_bc(0x80 | 0x40+3);       // DDRAM アドレスセットで，2 行目 ,4 列目にカーソルを移動
                                            // (DB7(1),DB6 ～ 0(ADD))
       write_data4_bc('1');                 // 文字の表示
       write_data4_bc('2');
       write_data4_bc('3');
       write_data4_bc('4');
       write_data4_bc('5');

       wait_ms(1000);
    }
}
```

データ書き込み時のビジーチェックは、while(read_cmnd4() & 0x80); で行います。LCD モジュールの初期化関数の中で、8 ビットモードの最後の命令 "write_cmnd8(0x20);" はビジーチェックが可能ですが、ここでは直前にウエイト (39μs) を挿入して対応しています。

実習 4.2.3a LCD の 1 行目に文字を表示し、それを読み出し（演算を加え)、2 行目に表示する

[解説]

4 ビットモードで（BF/AC）を読み出す関数を参考にデータを読み出す関数を作ります。

[解答]

リスト 4.2.3a にプログラム例を示します。

リスト 4.2.3a　LCD の 1 行目に文字を表示し、それを読み出し（演算を加え)、2 行目に表示する

```
/** main.c ***************************************************************************/
#include <mt_r300.h> // マイコンボード用ライブラリー
// VP1(b7: 未接続 ,b6:R/W(1: 読み出し /0: 書き込み ),b5:E(1: 許可 /0: 禁止 ,b4:RS(1: データ /0: 命令 ),b3 ～ 0
```

```
(DB7～4))
#define RD      0x40                    // R/W=1 ( 読み出し )
#define WT      0x00                    // R/W=0 ( 書き込み )
#define EN      0x20                    // E=1   ( 動作許可 )
#define DS      0x00                    // E=0   ( 動作禁止 )
#define DR      0x10                    // RS=1  ( データ )
#define IR      0x00                    // RS=0  ( 命令 )

void wait_ns(void)                      // 動作許可時間（実測値で約 710ns）
{
}

unsigned char read_cmnd4( void )        // 命令 ( ステータス ) を読み出す（4 ビットモードで）
{
    unsigned char cmnd;
    setup_VPD(0xf0);                    // VPD の下位 4 ビットを一時的に入力モードに設定
    out_VPD( IR | RD | DS );            // 動作禁止
    out_VPD( IR | RD | EN );            // 動作許可
//  wait_ns();                          // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    cmnd = in_VPD() << 4;               // 上位 4 ビットを読み出す ( シフト付 )
    out_VPD( IR | RD | DS );            // 動作禁止
//  wait_ns();                          // サイクルタイム (1200ns 以上 ) を確保 ( 可能なら省略 )
                                        // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( IR | RD | DS );            // 動作禁止，直前の命令に吸収する
    out_VPD( IR | RD | EN );            // 動作許可
//  wait_ns();                          // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    cmnd |= in_VPD() & 0x0f;            // 下位 4 ビットを読み出し、上位 4 ビットと合成する
    out_VPD( IR | RD | DS );            // 動作禁止
    setup_VPD(0xff);                    // VPD の下位 4 ビットを本来の出力モードに再設定
    return cmnd;
}

unsigned char read_data4( void )        // 文字を読み出す（4 ビットモードで，ビジーチェック付）
{
    unsigned char data;
    while(read_cmnd4() & 0x80);         // ビジー (BF=1(0x80)) のとき繰り返す ( 待つ )
    setup_VPD(0xf0);                    // VPD の下位 4 ビットを一時的に入力に設定
    out_VPD( DR | RD | DS );            // 動作禁止
    out_VPD( DR | RD | EN );            // 動作許可
//  wait_ns();                          // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    data = in_VPD() << 4;               // 上位 4 ビットを読み出す ( シフト付 )
    out_VPD( DR | RD | DS );            // 動作禁止
//  wait_ns();                          // サイクルタイム (1200ns 以上 ) を確保 ( 可能なら省略 )
                                        // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( DR | RD | DS );            // 動作禁止、直前の命令に吸収する
```

```
    out_VPD( DR | RD | EN );                // 動作許可
//  wait_ns();                              // イネーブル信号幅（230ns 以上 ）を確保（ 可能なら省略 ）
    data |= in_VPD() & 0x0f;                // 下位 4 ビットを読み出し、上位 4 ビットと合成する
    out_VPD( DR | RD | DS );                // 動作禁止
    setup_VPD(0xff);                        // VPD の下位 4 ビットを本来の出力に再設定
    return data;
}

void write_cmnd8(unsigned char cmnd)        // 命令を書き込む関数（8 ビットモードで，ウエイトなし）
{
    out_VPD( IR | WT | DS );                // 動作禁止
    out_VPD( IR | WT | EN | (cmnd >> 4) ); // 動作許可，上位 4 ビットを下位 4 ビットにシフトし書き込む
    wait_ns();                              // 230ns 以上待つ
    out_VPD( IR | WT | DS | (cmnd >> 4) ); // 動作禁止
}

void write_cmnd4_bc(unsigned char cmnd)     // 命令を書き込む（4 ビットモードで，ビジーチェック付 ）
{
    while(read_cmnd4() & 0x80);             // ビジー（BF=1(0x80)）のとき繰り返す（ 待つ ）
    out_VPD( IR | WT | DS );                // 動作禁止
    out_VPD( IR | WT | EN | (cmnd >> 4) ); // 動作許可，上位 4 ビットを書き込む（シフト付 ）
//  wait_ns();                              // イネーブル信号幅（230ns 以上 ）を確保（ 可能なら省略 ）
    out_VPD( IR | WT | DS | (cmnd >> 4) ); // 動作禁止
//  wait_ns();                              // サイクルタイム（1200ns 以上 ）を確保（ 可能なら省略 ）
                                            // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( IR | WT | DS );                // 動作禁止，直前の命令（ 命令付 ）に吸収する
    out_VPD( IR | WT | EN | (cmnd & 0x0f) );      // 動作許可，下位 4 ビットを書き込む
//  wait_ns();                              // イネーブル信号幅（230ns 以上 ）を確保（ 可能なら省略 ）
    out_VPD( IR | WT | DS | (cmnd & 0x0f) );      // 動作禁止
}

void write_data4_bc(char data)              // 文字を書き込む（4 ビットモードで，ビジーチェック付 ）
{
    while(read_cmnd4() & 0x80);             // ビジー（BF=1(0x80)）のとき繰り返す（ 待つ ）
    out_VPD( DR | WT | DS );                // 動作禁止
    out_VPD( DR | WT | EN | (data >> 4) ); // 動作許可，上位 4 ビットを書き込む（シフト付 ）
//  wait_ns();                              // イネーブル信号幅（230ns 以上 ）を確保（ 可能なら省略 ）
    out_VPD( DR | WT | DS | (data >> 4) ); // 動作禁止
//  wait_ns();                              // サイクルタイム（1200ns 以上 ）を確保（ 可能なら省略 ）
                                            // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( DR | WT | DS );                // 動作禁止，直前の命令（ データ付 ）に吸収する
    out_VPD( DR | WT | EN | (data & 0x0f) );      // 動作許可，下位 4 ビットを書き込む
//  wait_ns();                              // イネーブル信号幅（230ns 以上 ）を確保（ 可能なら省略 ）
    out_VPD( DR | WT | DS | (data & 0x0f) );      // 動作禁止
}
```

```
void initialize_lcd( void ) // LCD の初期化関数
{
    setup_VPD(0xff);         // VPD の初期化 (b7:(BZ),b6:(R/W),b5:(E),(b4:RS),b3 ～ 0:(DB7 ～ 4))
    wait_ms_itu2(45);        // LCD の準備時間 (VDD が 4.5V に達してから 45ms 以上待つ )
    write_cmnd8(0x30);       // まずファンクションセットで，8 ビットモードに設定する
                             // (DB5(1),DB4(DL=1:8 ビット ),DB3(N=0:1 行 ),DB2(F=0:5 × 7 ドット ))
    wait_us_itu2(4100);      // 4.1ms 以上待つ
    write_cmnd8(0x30);       // 同じファンクションセットを繰り返す (2 回目 )
                             // (DB5(1),DB4(DL=1:8 ビット ),DB3(N=0:1 行 ),DB2(F=0:5 × 7 ドット ))
    wait_us_itu2(100);       // 100μs 以上待つ
    write_cmnd8(0x30);       // 同じファンクションセットを繰り返す (3 回目 )
                             // (DB5(1),DB4(DL=1:8 ビット ),DB3(N=0:1 行 ),DB2(F=0:5 × 7 ドット ))
    wait_us_itu2(39);        // 39μs( ファンクションセット命令実行時間 ) 以上待つ
    write_cmnd8(0x20);       // ファンクションセットで，4 ビットモードに設定する
                             //  (DB5(1),DB4(DL=0:4 ビット ),DB3(N=0:1 行 ),DB2(F=0:5 × 7 ドット ))
    write_cmnd4_bc(0x28);    // ファンクションセットで，4 ビットモード，2 行表示に設定する
                    //  (DB5(1),DB4(DL=0:4 ビット ),DB3(N=1:2 行 ),DB2(F=*: 無効 (5 × 7 ドット )))
    write_cmnd4_bc(0x06);    // エントリーモードセットで，表示をシフトしないに設定する
                             // (DB2(1),DB1(I/D=*: 無効 ),DB0(S=0: 表示シフトしない ))
    write_cmnd4_bc(0x0c);    // 表示オン／オフで，表示をオンに，カーソルをオフ，点滅をオフにする
                             // (DB3(1),DB2(D=1: 表示オン ),DB1(C=0: カーソルオフ ),DB0(B=0: 点滅オフ ))
}

void main(void){
    int i;
    unsigned char d;

    initialize_lcd();                          // LCD の初期化
    while(1){
       // *** 1 行目に文字を表示する ***
       write_cmnd4_bc(0x0c);              // 表示オン／オフで，表示をオンに，カーソルをオフにする
                             // (DB3(1),DB2(D=1: 表示オン ),DB1(C=0: カーソルオフ ),DB0(B=0: 点滅オフ ))
       write_cmnd4_bc(0x01);              // 表示クリア (DB0(1))
       wait_ms(1000);

       write_cmnd4_bc(0x02);              // リターンホーム (DB1(1))
       write_data4_bc('A');               // 文字の表示
       write_data4_bc('B');
       write_data4_bc('C');
       write_data4_bc('D');
       write_data4_bc('E');
       write_data4_bc('F');
       write_data4_bc('G');               // 文字の表示
       write_data4_bc('H');
```

```
        write_data4_bc('I');

        // *** 1 行目の文字を読み出し，それを 2 行目に表示する ***
        write_cmnd4_bc(0x0f);        // 表示オン／オフで，表示をオン，カーソルと点滅をオンにする
                              //(DB3(1),DB2(D=1: 表示オン ),DB1(C=1: カーソルオン ),DB0(B=1: 点滅オン ))
        for(i = 0; i < 9; i++){
                write_cmnd4_bc(0x80 | 0x00+i);      // DDRAM アドレスセット (1 行目 ,i 列目 )(DB7(1),DB6
～ 0(ADD))
                d = read_data4();                   // 文字を読み出す
                write_cmnd4_bc(0x80 | 0x40+i);      // DDRAM アドレスセット (2 行目 ,i 列目 )(DB7(1),DB6
～ 0(ADD))
                write_data4_bc(d);                  // 文字をそのまま表示する ("ABCDEFGHI")
//              write_data4_bc(d + 0x20);           // 文字を演算して表示する ("abcdefghi")
//              write_data4_bc(d - 0x10);           // 文字を演算して表示する ("123456789")
                wait_ms(100);
        }
                wait_ms(1000);
    }
}
```

実習 4.2.3b 実習 4.2.3a のプログラムの関数名を標準的な関数名に変更し、必要な関数を追加する

[解説]

文字列を表示する関数は、文字を表示する関数を必要回数繰り返す方法で作成します。

[解答]

リスト 4.2.3b にプログラム例を示します。

リスト 4.2.3b　実習 4.2.3a のプログラムの関数名を標準的な関数名に変更し、必要な関数を追加する

```
/** main.c ********************************************************************************/
#include <mt_r300.h> // マイコンボード用ライブラリー
// VP1(b7: 未接続 ,b6:R/W(1: 読み出し /0: 書き込み ),b5:E(1: 許可 /0: 禁止 ,b4:RS(1: データ /0: 命令 ),b3 ～ 0
(DB7 ～ 4))
#define RD      0x40                        // R/W=1 ( 読み出し )
#define WT      0x00                        // R/W=0 ( 書き込み )
#define EN      0x20                        // E=1   ( 動作許可 )
#define DS      0x00                        // E=0   ( 動作禁止 )
#define DR      0x10                        // RS=1  ( データ )
#define IR      0x00                        // RS=0  ( 命令 )
#define ON      1
#define OFF     0

void wait_ns(void)                          // 動作許可時間（実測値で約 710ns）
{
```

```
}

unsigned char read_cmnd4_lcd( void )      // 命令 ( ステータス BF/AC) を読み出す (4 ビットモードで)
{
    unsigned char cmnd;
    setup_VPD(0xf0);                      // VPD の下位 4 ビットを一時的に入力モードに設定
    out_VPD( IR | RD | DS );              // 動作禁止
    out_VPD( IR | RD | EN );              // 動作許可
//  wait_ns();                            // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    cmnd = in_VPD() << 4;                 // 上位 4 ビットを読み出す ( シフト付 )
    out_VPD( IR | RD | DS );              // 動作禁止
//  wait_ns();                            // サイクルタイム (1200ns 以上 ) を確保 ( 可能なら省略 )
                                          // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( IR | RD | DS );              // 動作禁止, 直前の命令に吸収する
    out_VPD( IR | RD | EN );              // 動作許可
//  wait_ns();                            // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    cmnd |= in_VPD() & 0x0f;              // 下位 4 ビットを読み出し、上位 4 ビットと合成する
    out_VPD( IR | RD | DS );              // 動作禁止
    setup_VPD(0xff);                      // VPD の下位 4 ビットを本来の出力モードに再設定
    return cmnd;
}

unsigned char getbf_lcd( void )           // BF( ビジーフラグ ) を取り出す
{
    return ( read_cmnd4_lcd() & 0x80 ) >> 7;      // BF(1: ビジー, 0: レディ )
}

char getchar_lcd( void )                  // 文字を読み出す (4 ビットモードで, ビジーチェック付)
{
    unsigned char data;

    while( getbf_lcd() );                 // ビジー (BF=1) のとき繰り返す ( 待つ )
    setup_VPD(0xf0);                      // VPD の下位 4 ビットを一時的に入力に設定
    out_VPD( DR | RD | DS );              // 動作禁止
    out_VPD( DR | RD | EN );              // 動作許可
//  wait_ns();                            // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    data = in_VPD() << 4;                 // 上位 4 ビットを読み出す ( シフト付 )
    out_VPD( DR | RD | DS );              // 動作禁止
//  wait_ns();                            // サイクルタイム (1200ns 以上 ) を確保 ( 可能なら省略 )
                                          // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( DR | RD | DS );              // 動作禁止, 直前の命令に吸収する
    out_VPD( DR | RD | EN );              // 動作許可
//  wait_ns();                            // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    data |= in_VPD() & 0x0f;              // 下位 4 ビットを読み出し、上位 4 ビットと合成する
    out_VPD( DR | RD | DS );              // 動作禁止
```

```
    setup_VPD(0xff);                        // VPD の下位 4 ビットを本来の出力に再設定
    return data;
}

void write_cmnd8(unsigned char cmnd)                // 命令を書き込む（8 ビットモードで，ウエイトなし）
{
    out_VPD( IR | WT | DS );                        // 動作禁止
    out_VPD( IR | WT | EN | (cmnd >> 4) );          // 動作許可，上位を下位にシフトし書き込む
//  wait_ns();                                      // 230ns 以上待つ
    out_VPD( IR | WT | DS | (cmnd >> 4) );          // 動作禁止
}

void write_cmnd4_bc(unsigned char cmnd)             // 命令を書き込む（4 ビットモードで，ビジーチェック付）
{
    while( getbf_lcd() );                           // ビジー（BF=1）のとき繰り返す（待つ）
    out_VPD( IR | WT | DS );                        // 動作禁止
    out_VPD( IR | WT | EN | (cmnd >> 4) );          // 動作許可，上位 4 ビットを書き込む（シフト付）
//  wait_ns();                                  // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( IR | WT | DS | (cmnd >> 4) );          // 動作禁止
//  wait_ns();                                      // サイクルタイム（1200ns 以上）を確保（可能なら省略）
                                                // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( IR | WT | DS );                        // 動作禁止，直前の命令（命令付）に吸収する
    out_VPD( IR | WT | EN | (cmnd & 0x0f) );        // 動作許可，下位 4 ビットを書き込む
//  wait_ns();                                  // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( IR | WT | DS | (cmnd & 0x0f) );        // 動作禁止
}

void putchar_lcd(char data)                 // 文字を書き込む関数（4 ビットモードで，ビジーチェック付）
{
    while( getbf_lcd() );                           // ビジー（BF=1）のとき繰り返す（待つ）
    out_VPD( DR | WT | DS );                        // 動作禁止
    out_VPD( DR | WT | EN | (data >> 4) );          // 動作許可，上位 4 ビットを書き込む（シフト付）
//  wait_ns();                                  // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( DR | WT | DS | (data >> 4) );          // 動作禁止
//  wait_ns();                                      // サイクルタイム（1200ns 以上）を確保（可能なら省略）
                                                // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( DR | WT | DS );                        // 動作禁止，直前の命令（データ付）に吸収する
    out_VPD( DR | WT | EN | (data & 0x0f) );        // 動作許可，下位 4 ビットを書き込む
//  wait_ns();                                  // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( DR | WT | DS | (data & 0x0f) );        // 動作禁止
}

void initialize_lcd( void )         // LCD の初期化関数
{
    setup_VPD(0xff);        // VPD の初期化（b7:(BZ),b6:(R/W),b5:(E),(b4:RS),b3 ～ 0:(DB7 ～ 4)）
```

```
    wait_ms_itu2(45);         // LCD の準備時間（VDD が 4.5V に達してから 45ms 以上待つ）
    write_cmnd8(0x30);        // まずファンクションセットで，8 ビットモードに設定する
                              // (DB5(1),DB4(DL=1:8 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    wait_us_itu2(4100);       // 4.1ms 以上待つ
    write_cmnd8(0x30);        // 同じファンクションセットを繰り返す（2 回目）
                              // (DB5(1),DB4(DL=1:8 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    wait_us_itu2(100);        // 100 μ s 以上待つ
    write_cmnd8(0x30);        // 同じファンクションセットを繰り返す（3 回目）
                              // (DB5(1),DB4(DL=1:8 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    wait_us_itu2(39);         // 39 μ s（ファンクションセット命令実行時間）以上待つ
    write_cmnd8(0x20);        // ファンクションセットで，4 ビットモードに設定する
                              //  (DB5(1),DB4(DL=0:4 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
    write_cmnd4_bc(0x28);     // ファンクションセットで，4 ビットモード，2 行表示に設定する
                    //  (DB5(1),DB4(DL=0:4 ビット),DB3(N=1:2 行),DB2(F=*: 無効（5 × 7 ドット)))
    write_cmnd4_bc(0x06);     // エントリーモードセットで，表示をシフトしないに設定する
                              // (DB2(1),DB1(I/D=*: 無効),DB0(S=0: 表示シフトしない))
    write_cmnd4_bc(0x0c);     // 表示オン／オフで，表示をオン，カーソルをオフ，点滅をオフにする
                              // (DB3(1),DB2(D=1: 表示オン),DB1(C=0: カーソルオフ),DB0(B=0: 点滅オフ))
}

void clear_display(void)      // 表示をクリアする
{
    write_cmnd4_bc(0x01);
}

void return_home(void)        // ホームポジションに戻す
{
    write_cmnd4_bc(0x02);
}

void cursor_blink( unsigned char cursor, unsigned char blink)   // カーソルと点滅のオン / オフを設定
する
{
    write_cmnd4_bc( 0x0c | cursor<<1 | blink );
}                             // 表示オン / オフ制御（DB3(1: 固定),DB2(1: 表示),DB1( カーソル),DB0( 点滅))

void locate_cursor( unsigned char line_num, unsigned char column_num)  // (行，列）にカーソルを移
動する
{
    write_cmnd4_bc( 0x80 | ((line_num - 1) * 0x40 + (column_num - 1)));
}             // DDRAM アドレスセット（DB7(1: 固定),DB6 ～ 0(ADD))  ADD:(1 行目 :0x00 ～ ,2 行目 :0x40 ～ )

void prints_lcd(char *str)  // 文字列を表示する
{
     while(*str){                          // 文字列の最後（ヌル）になるまで繰り返す
```

```
        putchar_lcd(*str);
        str++;
     }
}

void main(void){
    int i;
    unsigned char d;

    initialize_lcd();                                    // LCD の初期化
    while(1){
       // *** 1 行目に文字を表示する ***
       cursor_blink(OFF, OFF);                           // カーソルをオフ，点滅をオフにする
       clear_display();                                  // 表示クリア
       wait_ms(1000);
       return_home();                                    // リターンホーム
       prints_lcd("ABCDEFGHIJKLMNOP");                   // 文字列の表示
       wait_ms(1000);

       // *** 1 行目の文字を読み出し，それを 2 行目に表示する ***
       cursor_blink(ON, ON);                             // カーソルをオン，点滅をオンにする
       for(i = 0; i < 16; i++){
              locate_cursor(1, i+1);                     // DDRAM アドレスセット（1 行目 ,i 列目 ）
              d = getchar_lcd();                         // 文字を読み出す
              locate_cursor(2, i+1);                     // DDRAM アドレスセット（2 行目 ,i 列目 ）
              putchar_lcd(d);                            // 文字をそのまま表示する（"ABCDEFGHI"）

//            putchar_lcd(d + 0x20);                     // 文字を演算して表示する（"abcdefghi"）
//            putchar_lcd(d - 0x10);                     // 文字を演算して表示する（"123456789"）
              wait_ms(100);
       }
              wait_ms(1000);
    }
}
```

実習 4.2.4　ビープ音（1kHz）を発生する

[解説]

音（振動数 f=1[KHz]，周期 t=1[ms]）を発生するために、0.5[ms] 毎に H/L のトグル信号を出力します。音の長さが ts[s] の時、for 文のループ回数は f × ts となります。

[解答]

リスト 4.2.4 にプログラム例を示します。

リスト 4.2.4　ビープ音（1KHz）を発生する

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>                          // マイコンボード用ライブラリー
void main (void)
{
    unsigned long i, is;
    float f, t, ts;
    setup_VPD(0xff);
    while(1){
       f = 1000;                                   // 音の振動数 1000[Hz]
       t = 1 / f * 1000000;                        // 音の周期 1/1000[s]=1000[μs]
       ts = 1;                                     // 音の長さ 1.0[ 秒 ]
       is = f * ts;                                // ループ回数 1000（周波数×音の長さ）
       for(i = 0; i < is; i++){
              out_VPD(0x80);                       // ビット 7:H 出力（LCD を同時使用しないで下さい）
              wait_us_itu2(t / 2);                 // 半周期
              out_VPD(0x00);                       // ビット 7:L 出力（LCD を同時使用しないで下さい）
              wait_us_itu2(t / 2);                 // 半周期
       }
       wait_ms(1000);
    }
}
```

4.3　ライブラリの作成（4 ビットモード）

4.2 と同様にマイコンボードと LCD ボードを接続します。"X:¥xxxx¥workspace" フォルダーにワークスペース、プロジェクト "lcd_4b_lib" を新規作成します。4.1、4.2 と同様に種々の設定をします。

これまでの実習で作成した関数をまとめたライブラリー（"mt_e502.c", "mt_e502.h"）を新規作成します。これらは、"mt_r300.c","mt_r300.h" などと同じフォルダー "X:¥xxxx¥workspace¥@library" に配置します。

続いて、"mt_r300.c" と同様に "mt_e502.c" もプロジェクトに追加します。

実習 4.3.1a　1 行目を下方向（2 行目）にシフト（コピー）する。

[解説]

ライブラリーの中に、文字を表示する関数を用いて、文字列（16 文字以内）を表示する関数を作成します。

[解答]

リスト 4.3.1a にプログラム例を示します。

リスト 4.3.1a　1 行目を 2 行目にシフト（コピー）する

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>                  // マイコンボード用ライブラリー
#include      <mt_e502.h>                  // LCD ボード用ライブラリー
void main(void){
    int i;
```

```
    unsigned char d;
    initialize_lcd();                       // LCD ボードの初期化
    while(1){
       clear_display();                     // 表示クリア
       wait_ms(1000);
       return_home();                       // リターンホーム　(1 行目、1 列目 )
       prints_lcd("Sunhayato Corp. ");      // 文字列の表示
       locate_cursor(2, 1);                 // カーソル位置設定 (2 行目、1 列目 )
       prints_lcd("VPort Laboratory");      // 文字列の表示
       wait_ms(2000);
       for(i = 1; i <= 16; i++){
              locate_cursor(1, i);
              d = getchar_lcd();                    // d に (1 行目 ,i 列目 ) の文字を保存
              locate_cursor(1, i);
              putchar_lcd(' ');                     // (1 行目、i 列目 ) の文字を消去 (コピー時は不要)
              locate_cursor(2, i);
              putchar_lcd(d);                       // (2 行目、i 列目 ) に d を表示する
              wait_ms(200);                         // 適宜調整
       }
       wait_ms(2000);
    }
}
```

```
/** mt_e502.c ***************************************************************************/
#include      <mt_r300.h>                   // mt_r300 用ライブラリー
// VP1(b6:R/W(1: 読出 /0: 書込 ),b5:E(1: 許可 /0: 禁止 ,b4:RS(1: データ /0: 命令 ),b3 ～ 0(DB7 ～ 4))
#define RD      0x40                        // R/W=1 ( 読み出し )
#define WT      0x00                        // R/W=0 ( 書き込み )
#define EN      0x20                        // E=1   ( 動作許可 )
#define DS      0x00                        // E=0   ( 動作禁止 )
#define DR      0x10                        // RS=1  ( データ )
#define IR      0x00                        // RS=0  ( 命令 )
void wait_ns(void)                          // 実測値で約 710ns
{
}
unsigned char read_cmnd4_lcd( void )        // 命令 ( ステータス BF/AC) を読み出す (4 ビットモードで)
{
    unsigned char cmnd;
    setup_VPD(0xf0);                        // VPD の下位 4 ビットを一時的に入力モードに設定
    out_VPD( IR | RD | DS );                // 動作禁止
    out_VPD( IR | RD | EN );                // 動作許可
//  wait_ns();                              // イネーブル信号幅 (230ns 以上 ) を確保 ( 可能なら省略 )
    cmnd = in_VPD() << 4;                   // 上位 4 ビットを読み出す ( シフト付 )
    out_VPD( IR | RD | DS );                // 動作禁止
//  wait_ns();                              // サイクルタイム (1200ns 以上 ) を確保 ( 可能なら省略 )
```

```
                                        // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( IR | RD | DS );            // 動作禁止，直前の命令に吸収する
    out_VPD( IR | RD | EN );            // 動作許可
//  wait_ns();                          // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    cmnd |= in_VPD() & 0x0f;            // 下位 4 ビットを読み出し、上位 4 ビットと合成する
    out_VPD( IR | RD | DS );            // 動作禁止
    setup_VPD(0xff);                    // VPD の下位 4 ビットを本来の出力モードに再設定
    return cmnd;
}

unsigned char getbf_lcd( void )                 // BF( ビジーフラグ ) を読み出す
{
    return ( read_cmnd4_lcd() & 0x80 ) >> 7;    // BF(1: ビジー，0: レディ )
}

unsigned char getac_lcd( void )         // AC( アドレスカウンタ ) 値を読み出す
{
    return read_cmnd4_lcd() & 0x7f;     // AC(b6 ～ 0 ビット )
}

char getchar_lcd( void )                // 文字を読み出す（4 ビットモードで，ビジーチェック付）
{
    unsigned char data;
    while( getbf_lcd() );               // ビジー（BF=1）のとき繰り返す（待つ）
    setup_VPD(0xf0);                    // VPD の下位 4 ビットを一時的に入力に設定
    out_VPD( DR | RD | DS );            // 動作禁止
    out_VPD( DR | RD | EN );            // 動作許可
//  wait_ns();                          // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    data = in_VPD() << 4;               // 上位 4 ビットを読み出す（シフト付）
    out_VPD( DR | RD | DS );            // 動作禁止
//  wait_ns();                          // サイクルタイム（1200ns 以上）を確保（可能なら省略）
                                        // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( DR | RD | DS );            // 動作禁止，直前の命令に吸収する
    out_VPD( DR | RD | EN );            // 動作許可
//  wait_ns();                          // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    data |= in_VPD() & 0x0f;            // 下位 4 ビットを読み出し、上位 4 ビットと合成する
    out_VPD( DR | RD | DS );            // 動作禁止
    setup_VPD(0xff);                    // VPD の下位 4 ビットを本来の出力に再設定
    return data;
}

void write_cmnd8(unsigned char cmnd)    // 命令を書き込む（8 ビットモードで，ウエイトなし）
{
    out_VPD( IR | WT | DS );            // 動作禁止
    out_VPD( IR | WT | EN | (cmnd >> 4) ); // 動作許可，上位 4 ビットを下位 4 ビットにシフトし書き込む
```

```
//  wait_ns();                              // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( IR | WT | DS | (cmnd >> 4) ); // 動作禁止
}

void write_cmnd4_bc(unsigned char cmnd)    // 命令を書き込む（4 ビットモードで，ビジーチェック付）
{
    while( getbf_lcd() );                  // ビジー（BF=1）のとき繰り返す（待つ）
    out_VPD( IR | WT | DS );               // 動作禁止
    out_VPD( IR | WT | EN | (cmnd >> 4) ); // 動作許可，上位 4 ビットを書き込む（シフト付）
//  wait_ns();                              // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( IR | WT | DS | (cmnd >> 4) ); // 動作禁止
//  wait_ns();                              // サイクルタイム（1200ns 以上）を確保（可能なら省略）
                                           // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( IR | WT | DS );               // 動作禁止，直前の命令（命令付）に吸収する
    out_VPD( IR | WT | EN | (cmnd & 0x0f) );      // 動作許可，下位 4 ビットを書き込む
//  wait_ns();                              // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( IR | WT | DS | (cmnd & 0x0f) );      // 動作禁止
}

void putac_lcd(unsigned char ac)           // AC（アドレスカウンタ）に DDRAM アドレスを書き込む関数
{
    write_cmnd4_bc( 0x80 | ac);            // DDRAM アドレスセット（DB7(1),DB6 ～ 0(ADD)）
}

void putchar_lcd(char data)                // 文字を書き込む（4 ビットモードで，ビジーチェック付）
{
    while( getbf_lcd() );                  // ビジー（BF=1）のとき繰り返す（待つ）
    out_VPD( DR | WT | DS );               // 動作禁止
    out_VPD( DR | WT | EN | (data >> 4) ); // 動作許可，上位 4 ビットを書き込む（シフト付）
//  wait_ns();                              // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( DR | WT | DS | (data >> 4) ); // 動作禁止
//  wait_ns();                              // サイクルタイム（1200ns 以上）を確保（可能なら省略）
                                           // VPort のアクセスタイムが約 1000ns のためウエイトを省略
//  out_VPD( DR | WT | DS );                      // 動作禁止，直前の命令（データ付）に吸収する
    out_VPD( DR | WT | EN | (data & 0x0f) );      // 動作許可，下位 4 ビットを書き込む
//  wait_ns();                              // イネーブル信号幅（230ns 以上）を確保（可能なら省略）
    out_VPD( DR | WT | DS | (data & 0x0f) );      // 動作禁止
}

void initialize_lcd( void )        // LCDの初期化関数(LCDのファンクションセット等とVPDのセットアップ)
{
    setup_VPD(0xff);         // VPD の初期化（b7:(BZ),b6:(R/W),b5:(E),(b4:RS),b3 ～ 0:(DB7 ～ 4)）
    wait_ms_itu2(45);        // LCD の準備時間（VDD が 4.5V に達してから 45ms 以上待つ）
    write_cmnd8(0x30);       // まずファンクションセットで，8 ビットモードに設定する
                             // (DB5(1),DB4(DL=1:8 ビット),DB3(N=0:1 行),DB2(F=0:5 × 7 ドット))
```

```
    wait_us_itu2(4100);      // 4.1ms 以上待つ
    write_cmnd8(0x30);       // 同じファンクションセットを繰り返す（2 回目 ）
    wait_us_itu2(100);       // 100 μ s 以上待つ
    write_cmnd8(0x30);       // 同じファンクションセットを繰り返す（3 回目 ）
    wait_us_itu2(39);        // 39 μ s( ファンクションセット命令実行時間 ) 以上待つ
    write_cmnd8(0x20);       // ファンクションセットで，4 ビットモードに設定
                      //  (DB5(1),DB4(DL=0:4 ビットモード ),DB3(N=0:1 行 ),DB2(F=0:5 × 7 ドット ))
    write_cmnd4_bc(0x28);    // ファンクションセットで，4 ビットモード，2 行表示に設定
                      //  (DB5(1),DB4(DL=0:4 ビットモード ),DB3(N=1:2 行 ),DB2(F=*: 無効 (5 × 7 ドット )))
    write_cmnd4_bc(0x06);    // エントリーモードセットで，カーソル右移動，表示シフトしないに設定
                             // (DB2(1),DB1(I/D=1: カーソル右移動 ),DB0(S=0: 表示シフトしない ))
    write_cmnd4_bc(0x0c);    // 表示オン／オフで，表示オン，カーソルオフ，点滅オフに設定
                             // (DB3(1),DB2(D=1: 表示オン ),DB1(C=0: カーソルオフ ),DB0(B=0: 点滅オフ ))
}

void clear_display(void)     // 表示をクリアする
{
    write_cmnd4_bc(0x01);    // 表示クリア (DB0(1))
}

void return_home(void)       // ホームポジションにカーソルを戻す
{
    write_cmnd4_bc(0x02);    // リターンホーム (DB1(1))
}

void cursor_blink( unsigned char cursor, unsigned char blink)  // カーソル表示と点滅を ON/OFF する
{
    write_cmnd4_bc( 0x0c | cursor << 1 | blink );               //（表示オン／オフ）
}                                          // (DB3(1),DB2(1: 表示 ),DB1( カーソル ),DB0( 点滅 ))

void locate_cursor( unsigned char line_num, unsigned char column_num ) // 指定の行，列にカーソルを
移動する
{
    write_cmnd4_bc( 0x80 | ((line_num-1)*0x40 + (column_num-1)));      // DDRAM アドレスセット
}                                                                     // (DB7(1),DB6 ～ 0(ADD))
int prints_lcd(char *str)                                             // 文字列を表示する
{
    int cnt = 0, err_flag = 0;
    while(*str && err_flag == 0){
        putchar_lcd(*str);
        str++;
       if(++cnt >= 16)
              err_flag = 1;
    }
    return err_flag;
```

```
}
```

```
/** mt_e502.h ***************************************************************************/
unsigned char getbf_lcd( void );
unsigned char getac_lcd( void );
char getchar_lcd( void );
void putac_lcd( unsigned char );
void putchar_lcd( char );void initialize_lcd( void );
void clear_display( void );
void return_home( void );
void cursor_blink( unsigned char, unsigned char );
void locate_cursor( unsigned char, unsigned char );
int prints_lcd( char * );
```

実習 4.1.3b 実習 4.3.1a を参考に、上下方向にシフトする関数を作成する

[解答]

リスト 4.3.1b にプログラム例を示します。ライブラリーは追加分のみです。それぞれに追加してください。

リスト 4.3.1b　上下方向にシフトする関数を作成します

```
/** main.c  ****************************************************************************/
#include      <mt_r300.h>                 // マイコンボード用ライブラリー
#include      <mt_e502.h>                 // LCD ボード用ライブラリー
void main(void){
    initialize_lcd();                     // LCD ボードの初期化
    while(1){
       clear_display();                   // 表示クリア
       wait_ms(1000);
       return_home();                     // リターンホーム　(1 行目、1 列目 )
       prints_lcd("Sunhayato Corp. ");    // 文字列の表示
       locate_cursor(2, 1);               // カーソル位置設定 (2 行目、1 列目 )
       prints_lcd("VPort Laboratory");    // 文字列の表示
       wait_ms(2000);
//     shift_down();                      // 下方向にシフト
       shift_up();                        // 上方向にシフト
       wait_ms(2000);
    }
}
```

```
/** mt_e502.c (追加分のみ) ***************************************************************/
void shift_down(void)                     // 1 行目を下方向 (2 行目 ) にシフトする
{
    int i;
    unsigned char d;
    for(i = 1; i <= 16; i++){
```

```
        locate_cursor(1, i);
        d = getchar_lcd();                  // d に（1 行目、i 列目 ）の文字を保存
        locate_cursor(1, i);
        putchar_lcd(' ');                   // その場所の文字を消去
        locate_cursor(2, i);
        putchar_lcd(d);                     // （2 行目、i 列目 ）に d を表示
        wait_ms(10);                        // 適宜調整
    }
}
void shift_up(void)                         // 2 行目を上方向（1 行目 ）にシフトする
{
    int i;
    unsigned char d;
    for(i = 1; i <= 16; i++){
        locate_cursor(2, i);
        d = getchar_lcd();                  // d に（2 行目、i 列目 ）の文字を保存
        locate_cursor(2, i);
        putchar_lcd(' ');                   // その場所の文字を消去
        locate_cursor(1, i);
        putchar_lcd(d);                     // （1 行目、i 列目 ）に d を表示する
        wait_ms(10);                        // 適宜調整
    }
}
```

```
/** mt_e502.h（追加分のみ）*****************************************************************/
void shift_down(void);
void shift_up(void);
```

実習 4.3.2a 文字列を表示し、それを左右方向にローテーションする

[解答]

リスト 4.3.2a にプログラム例を示します。

リスト 4.3.2a　文字列を表示し、左右方向にローテーションする

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>                   // マイコンボード用ライブラリー
#include      <mt_e502.h>                   // LCD ボード用ライブラリー
void main(void){
    int i, j;
    char d, d1;
    initialize_lcd();                       // LCD ボードの初期化
    clear_display();
    wait_ms(1000);
    return_home();
    prints_lcd("Sunhayato Corp. ");
```

```
    locate_cursor(2, 1);
    prints_lcd("VPort Laboratory");
    wait_ms(1000);
    while(1){
       for(i=1; i<=2; i++){                        // 左方向にローテーション
              locate_cursor( i, 1 );
              d1 = getchar_lcd();                  // c1 に（i 行目、1 列目 )( 左端 ) の文字を保存
              for(j=2; j<=16; j++){
                     locate_cursor( i, j );
                     d = getchar_lcd();            // c に（i 行目、j 列目 ) の文字を保存
                     locate_cursor( i, j );
                     putchar_lcd(' ');             // (i 行目、j 列目 ) の文字を消去
                     locate_cursor( i, j-1 );
                     putchar_lcd(d);               // (i 行目、j-1 列目 ) に c を表示
              }
              locate_cursor( i, 16 );
              putchar_lcd(d1);                     // (i 行目、16 列目 )( 右端 ) に c1 を表示
       }
       wait_ms(1000);
    }
}
```

実習 4.3.2b 実習 4.3.2a を参考に、左右方向にローテーションする関数を作成する

[解答]

リスト 4.3.2b にプログラム例を示します。

リスト 4.3.2b　左右方向にシフトする関数を作成します

```
/** main.c  ***************************************************************************/
#include      <mt_r300.h>          // マイコンボード用ライブラリー
#include      <mt_e502.h>          // LCD ボード用ライブラリー
void main(void){
    initialize_lcd();              // LCD ボードの初期化
    clear_display();
    wait_ms(1000);
    return_home();
    prints_lcd("Sunhayato Corp. ");
    locate_cursor(2, 1);
    prints_lcd("VPort Laboratory");
    wait_ms(1000);
    while(1){
//     rotation_left();                          // 左方向にローテーション
       rotation_right();                         // 右方向にローテーション
       wait_ms(1000);
    }
```

```
}
```

```
/** mt_e502.c (追加分のみ) ****************************************************************/
void rotation_left(void)                          // 表示文字を左方向にローテーションする
{
    int i, j;
    char d, d1;
    for(i=1; i<=2; i++){
       locate_cursor( i, 1 );
       d1 = getchar_lcd();                        // d1 に i 行目、1 列目 ( 左端 ) の文字を保存
       for(j=2; j<=16; j++){
              locate_cursor( i, j );
              d = getchar_lcd();                  // d に i 行目、j 列目の文字を保存
              locate_cursor( i, j );
              putchar_lcd(' ');                   // その場所の文字を消去
              locate_cursor( i, j-1 );
              putchar_lcd(d);                     // i 行目、j-1 列目に d を表示
       }
       locate_cursor( i, 16 );
       putchar_lcd(d1);                           // i 行目、16 列目 ( 右端 ) に d1 を表示
    }
}
void rotation_right(void)                         // 表示文字を右方向にローテーションする
{
    int i, j;
    char d, d16;
    for(i=1; i<=2; i++){
       locate_cursor( i, 16 );
       d16 = getchar_lcd();                       // d16 に i 行目、16 列目 ( 右端 ) の文字を保存
       for(j=15; j>=1; j--){
              locate_cursor( i, j );
              d = getchar_lcd();                  // d に i 行目、j 列目の文字を保存
              locate_cursor( i, j );
              putchar_lcd(' ');                   // その場所の文字を消去
              locate_cursor( i, j+1 );
              putchar_lcd(d);                     // i 行目、j+1 列目に d を表示
       }
       locate_cursor( i, 1 );
       putchar_lcd(d16);                          // i 行目、1 列目 ( 左端 ) に d16 を表示
    }
}
```

```
/** mt_e502.h (追加分のみ) ****************************************************************/
void rotation_left(void);
void rotation_right(void);
```

実習 4.3.3　LCD に整数（long 型）を表示する（引数：フィールド幅 , 整数）

[解説]

整数を 10 で割って商と余りを求めます。余りは 1 の桁の値になります。次に商に対して同じ演算を繰り返すと、順次 10 の桁、100 の桁の値が求められます。負数は－ 1 をかけて正数にして同じ演算をします。数字（0 ～ 9）を表示するときは '0'（0x30）を加算して文字コード（0x30 ～ 0x39）に変換します。

この中で pow() 関数を使いますので、main() 関数内に "#include <math.h>" を追加します。

プロジェクトに標準ライブラリー "math.c" を追加するために、次のとおりを設定します。

（1）[ビルド]、[H8S、H8/300 Standerd TOOLchain] を選択し、"H8S、H8/300 Standerd TOOLchain" ウインドウに切り替えます。

（2）[標準ライブラリー] タグをクリックし、[カテゴリ] で " 標準ライブラリー " を選択します。

（3）必要なライブラリー "math.c" にチェックし、[OK] を押下します。

[解答]

リスト 4.3.3 にプログラム例を示します。

リスト 4.3.3　LCD に整数 (long 型) を表示する（引数：フィールド幅 , 整数）

```
/** main.c  ********************************************************************************/
#include      <math.h>                    // 数値計算用ライブラリー
#include      <mt_r300.h>                 // マイコンボード用ライブラリー
#include      <mt_e502.h>                 // LCD ボード用ライブラリー
void wait_shift_up()
{
    wait_ms(2000);
    shift_up();
    wait_ms(500);
    locate_cursor(2, 1);
}
void main(void){
    initialize_lcd();
    while(1){
       clear_display();
       wait_ms(1000);
       locate_cursor(2, 1);
       prints_lcd("printi_lcd      ");
       wait_shift_up();
       printi_lcd( 16,  12345 );
       wait_shift_up();
       printi_lcd( 16,  -12345 );
       wait_shift_up();
       printi_lcd( 10,  2147483647 );            // long 型の最大値
       wait_shift_up();
       printi_lcd( 11,  -2147483648 );           // long 型の最小値
       wait_shift_up();
```

```
        printi_lcd(  5,  1 );
        wait_shift_up();
        printi_lcd(  5,  -1 );
        wait_shift_up();
        printi_lcd(  5,  0 );
        wait_shift_up();
        printi_lcd( 10,  4294967294 );          // long 型の範囲外のため負数 -2 に化けます
        wait_shift_up();
        printi_lcd(  9,  1234567890 );          // 指定桁オーバーフローエラー（9 文字まで ）
        wait_shift_up();
    }
}
```

```
/** mt_e502.c（追加分のみ）*****************************************************************/
#include <math.h>                               // 数値計算用ライブラリー ( プログラムの先頭に配置 )

int printi_lcd( int field_w, long idat)         // 整数を表示する（フィールド幅，整数（long 型 ））
{
    char lcd_out_buf[17];                       // 出力バッファー
    int  i, err_flag = 0;
    long syo;
    char      err_msg1[] = "Over flow error ";
    if(-pow(10.0,(double)(field_w-1)) < idat && idat < pow(10.0,(double)field_w)){
       if( idat >= 0 ){                                         // 正数の場合
              for(i = field_w - 1; i >= 0; i--){
                     syo = idat / 10;                           // 10 で割った商を求める
                     lcd_out_buf[i] = idat - syo * 10 + '0';    // 10 で割った余りを文字に変換し代入
                     idat = syo;                                // 商を次の値とする
              }
              i = 0;
              while( lcd_out_buf[i] == '0' && i < field_w-1 ){
                     lcd_out_buf[i] = ' ';                      // 不要なゼロを消去する
                     i++;
              }
       }
       else {                                                   // 負数の場合
              for(i = field_w - 1; i > 0; i--){
                     syo = idat / 10;                           // 10 で割った商を求める
                     lcd_out_buf[i] = -(idat-syo*10)+'0'; // 10 で割った余り ( 絶対値 ) を文字変換
                     idat = syo;                                // 商を次の値とする
              }
              lcd_out_buf[0] = '-';
              i = 1;
              while( lcd_out_buf[i] == '0' && i < field_w-1 ){
                     lcd_out_buf[i] = lcd_out_buf[i-1]          // 負号を下の桁にシフトする
```

```
                    lcd_out_buf[i-1] = ' ';                    // 不要な負号を消去する
                    i++;
                }
        }
        lcd_out_buf[field_w] = '\0';                          // 出力バッファ長をフィールド長に合わせる
        prints_lcd(lcd_out_buf);                              // 文字列の表示
    }
    else{                                                     // オーバーフローの場合
        err_msg1[ field_w ] = '\0';                           // メッセージ長をフィールド長に合わせる
        prints_lcd(err_msg1);                                 // エラーメッセージの表示
        err_flag = 1;
    }
    return    err_flag;
}
```

```
/** mt_e502.h（追加分のみ）*************************************************************/
int printi_lcd( int, long );
```

実習 4.3.4　ビープ音を発生する

[解答]

リスト 4.3.4 にプログラム例を示します。

リスト 4.3.3　ビープ音 (1KHz) を発生する

```
/** main.c  ****************************************************************************/
#include      <mt_r300.h>                          // マイコンボード用ライブラリー
#include      <mt_e502.h>                          // LCD ボード用ライブラリー
void main (void)
{
    while(1){
        beep(0.5);                                 // 1KHz のビープ音 (0.5[s])
        wait_ms(500);
    }
}
```

```
/** mt_e502.c（追加分のみ）*************************************************************/
void beep( float ts )
{
    unsigned long i, is;
    float f, t;
    setup_VPD(0xff);
    f = 1000;                                  // 音の振動数 1000[Hz]
    t = 1 / f * 100000;                        // 音の周期 1/1000[s]*100000=100[10μs]
    is = f * ts;                               // ループ回数（周波数×音の長さ）
    for(i = 0; i < is; i++){
```

```
        out_VPD(0x80);                          // ビット 7:H 出力（同時に LCD に表示しないでください）
        wait_us(t / 2);                         // 半周期
        out_VPD(0x00);                          // ビット 7:L 出力（同時に LCD に表示しないでください）
        wait_us(t / 2);                         // 半周期
    }
}
```

```
/** mt_e502.h（追加分のみ）*************************************************************/
void beep( float );
```

5. おわりに

ここまでの実習により、8 ビットモードでは簡単に制御プログラムを作成できたと思います。十分理解を深めてから 4 ビットモードに移行することで、複雑なプログラムも自然に習得できたと思います。このボードで、R/W 信号の切り換えにより文字の書き込みだけでなく読み出しもできました。状態（BF/AC）を読み出すことで、ビジーチェックもできました。評価用 LED により、RS、E、R/W の信号レベル（H/L）の変化や文字のアスキーコードを視認できました。

ここでは、文字列や整数の表示などの基本的な関数を作成しました。これらを進化させることで、ネイティブな C 言語でお馴染みの書式付出力関数（printf_lcd("%8.3", a); など）も作成できます。キースキャンボード MT-E503 と組み合わせることで電卓なども構築できます。これらの応用的な実習は別のテキストに譲ります。

マイコントレーニング基板　MT-R300
トレーニングテキスト　LCD 基板編

発行日　　2012-04-20　Rev1.02
発　行　　サンハヤト株式会社
〒 170-0005　東京都豊島区南大塚 3 丁目 40 番 1 号

©2010 Sunhayato Corp. All rights reserved.

SG107014